Higher-order languages : dualities and bisimulation enhancements

Jean-Marie Madiot

N° attribué par la bibliothèque: 2015ENSL0988

- ÉCOLE NORMALE SUPÉRIEURE DE LYON -
Laboratoire de l'Informatique du Parallélisme - UMR5668 - LIP

- UNIVERSITÀ DI BOLOGNA -
Dipartimento di Informatica - Scienza e Ingegneria - DISI

THÈSE

en vue d'obtenir les grades de

Docteur de l'Université de Lyon, délivré par l'École Normale Supérieure de Lyon

Spécialité : Informatique

et

Dottore di Ricerca dell'Università di Bologna

au titre de l'École Doctorale Informatique et Mathématiques

présentée et soutenue publiquement le 31 mars 2015 par

Jean-Marie MADIOT

Higher-order languages: dualities and bisimulation enhancements

Directeurs de thèse : Daniel HIRSCHKOFF
Davide SANGIORGI

Devant la commission d'examen formée de :

Pierre-Louis	CURIEN	*Examinateur*
Daniel	HIRSCHKOFF	*Examinateur*
Vasileios	KOUTAVAS	*Rapporteur*
Davide	SANGIORGI	*Examinateur*
Daniele	VARACCA	*Rapporteur*
Björn	VICTOR	*Rapporteur*

Résumé en français

Langages d'ordre supérieur : dualités et techniques de bisimulation

Les comportements des processus concurrents peuvent être exprimés en utilisant des *calculs de processus*, des langages formels simples qui permettent de démontrer des résultats mathématiques précis sur les comportements et les interactions entre processus (Chapitre 1).

Un exemple simple mais expressif d'un tel langage est CCS (Calculus of Communicating Systems), dont on détaille l'étude des comportements. Un autre exemple répandu est le pi-calcul, qui est encore plus expressif grâce à un mécanisme de communication de canaux. Ce dernier autorise également l'instauration d'un système de types, qui permet de raffiner le calcul aux processus qui évoluent dans des environnements plus contraints. Le pi-calcul permet aussi d'encoder le lambda-calcul, qui lui-même capture l'essence du calcul séquentiel (Chapitre 2).

Plusieurs calculs de processus se déclinent avec des niveaux variables de complexité. Une propriété intéressante de certains de ces calculs est ce qu'on appelle la symétrie : certains calculs, comme CCS, le pi-calcul à mobilité interne ou les calculs dits "de fusions", sont invariants par une opération de dualité entre émission et réception (Sections 3.1 et 3.2). La première contribution de cette thèse se consacre à l'utilisation de cette dualité dans le pi-calcul (Chapitre 3). Le pi-calcul n'étant pas lui-même stable par dualité, on construit par saturation une extension conservatrice du pi-calcul typé, qui elle est stable. Grâce à cet outil, on établit une correspondance entre deux encodages du lambda-calcul dans le pi-calcul qui sont pourtant d'apparences différentes (Section 3.3).

Cette correspondance amène à la construction d'un nouveau calcul symétrique typé. On se pose alors la question de la possibilité d'utiliser des systèmes de types dans les calculs symétriques existants. La suite du chapitre se concentre sur cette question et se termine par une réponse négative : on établit un théorème d'impossibilité, qui spécifie que les calculs de fusions ne peuvent pas supporter l'ajout de disciplines de types, sauf dans des cas triviaux (Section 3.4).

En étudiant la preuve de ce théorème d'impossibilité, on découvre une contrainte fondamentale de ces calculs qui empêche l'utilisation des types : les processus induisent une relation d'équivalence entre les canaux de communication, qui est particulièrement évidente dans la variante du calcul des fusions "explicites". En relâchant cette contrainte pour que les canaux de communication soient au lieu de cela sujets à une relation de pré-ordre, on obtient assez directement un nouveau calcul, dans le style des calculs de fusions, qu'on appelle "pi-calcul avec pré-ordres" et dont on étudie les propriétés dans la suite (Chapitre 4).

Comme dans le calcul des fusions explicites, un opérateur du calcul appelé "arc" représente la relation induite sur les noms, en imposant à deux noms d'être reliés dans la relation globale de pré-ordre. Ces arcs agissent grâce à un système de réécriture très simple, et définissent directement le comportement du calcul (Section 4.1). On récupère alors dans le pi-calcul avec pré-ordres la même notion de types que dans le pi-calcul (Section 4.2).

La sémantique par réécriture est "avide", dans le sens où les arcs peuvent réagir avec n'importe quel préfixe à n'importe quel moment. On présente une autre sémantique plus paresseuse, qui n'utilise les arcs qu'au moment de la communication (Section 4.3). Elle décrit des étapes de réduction plus traditionnelles, ce qui permet des propriétés plus intéressantes et des correspondances plus fines avec les calculs existants.

Le calcul fait également preuve d'une capacité à gérer les noms privés, cette dernière ayant été perdue à dessein lors de l'élaboration des calculs de fusions à partir du pi-calcul original (Section 4.4). Cette capacité à créer de nouveaux noms privés est largement utilisée dans les implémentations, ce qui s'ajoute au fait le calcul est muni d'un système de types.

On décrit ensuite la sémantique comportementale paresseuse du pi-calcul avec pré-ordres et ses spécificités, à travers l'élaboration d'un système de transitions étiquetées, une bisimulation, des preuves techniques de résultats usuels de congruence, dont certaines sont formalisées en utilisant coq (Section 4.5). Une deuxième caractérisation est donnée sous la forme d'une liste de lois comportementale qui capture complètement la sémantique. Enfin, cette axiomatisation peut être adaptée pour obtenir, pour la première fois, une axiomatisation du calcul des fusions explicites (Section 4.6).

Le pi-calcul avec pré-ordres est suffisamment expressif pour encoder le pi-calcul et les calculs de fusions, avec une correspondance particulièrement fine pour les processus asynchrones (Section 4.7). Enfin, on voit que l'encodage du pi-calcul induit une restriction syntaxique naturelle sur les calculs avec fusions et pré-ordres. On peut montrer que les calculs coïncident sur le domaine de cette restriction, et qu'on peut même y ajouter une discipline de types, découvrant un autre moyen de recouvrer l'utilisation des types dans les calculs de fusions (Section 4.8).

On s'intéresse ensuite (Chapitre 5) aux méthodes de preuve pour la sémantique comportementale des processus, plus précisément à la méthode de la bisimulation. Cette dernière permet de capturer efficacement les équivalences désirées dans plusieurs langages d'ordre supérieur, comme le pi-calcul d'ordre supérieur et le lambda-calcul. Une preuve de bisimulation est habituellement constituée d'une première partie qui construit une relation (qui doit contenir les objets à prouver égaux), et d'une deuxième qui s'occupe de prouver que cette relation est une bisimulation. La deuxième partie de la preuve nécessite habituellement d'étendre la relation de la première partie, et ainsi de suite jusqu'à obtenir une relation qui est une bisimulation. Une partie de ce travail d'itération est assez administrative et répétitive, et peut être en fait factorisée pour simplifier les relations de façon radicale, parfois décisive pour compléter la preuve, en utilisant les techniques de bisimulation "modulo" (par exemple, une bisimulation "modulo contexte" peut être beaucoup plus petite qu'une bisimulation stricte). Pous et Sangiorgi ont dévelopé une théorie générale des bisimulations modulo, qui fournit un cadre dans lequel utiliser ces techniques se fait de façon modulaire : quand deux techniques modulo sont prouvées, leur combinaison, par exemple par l'union ou la composition, est automatiquement dérivée (Section 5.1).

En revanche cette théorie s'applique seulement dans le cadre de systèmes exprimés de façon relativement simple, comme pour les automates ou encore des calculs de processus comme CCS. Les techniques de bisimulations modulo pour des langages d'ordre supérieur (comme le pi-calcul, le lambda-calcul et leurs variantes) doivent malheureusement être adaptées à la main à partir des preuves de la théorie générale, et présentent l'inconvénient sérieux de requérir une nouvelle preuve pour chaque combinaison de techniques, contrairement à ce qu'il se passe dans la théorie générale.

Cette thèse contribue à adapter dans des langages d'ordre supérieur la théorie générale des techniques modulo. L'approche utilisée est d'établir une correspondance exacte entre un langage d'ordre supérieur et un système de transitions étiquetées du premier ordre construit à cet effet, tel que la bisimulation induite sur le système du premier ordre coïncide avec l'équivalence comportementale désirée dans le langage d'ordre supérieur. On applique cette méthode pour le pi-calcul, en utilisant une notion de processus nommés (Section 5.2), puis sur le lambda-calcul, en se basant sur une notion de bisimulation spécialisée, dite environnementale (Section 5.3). On explique ensuite comment gérer un langage avec des primitives de programmation impérative (Section 5.4) et d'autres langages, comme le pi-calcul d'ordre supérieur où les objets envoyés sont eux-mêmes des processus (Section 5.5).

Dans chacun de ces cas, grâce à la description au premier ordre des langages, on hérite toutes les techniques génériques de la théorie. On prouve aussi les techniques "modulo contexte" qui sont spécifiques à chacun de ces calculs, et qui forment une base solide pour établir des preuves de bisimulation. Les conditions pour que ces techniques spécifiques soient correctes ne sont pas toujours satisfaites, et peuvent dépendre de la manière dont on a construit le système de transitions du premier ordre. Cette étude s'efforce de guider le lecteur dans la marche à suivre pour construire des systèmes de transitions qui admettent ces techniques, pour la gestion d'autres langages d'ordre supérieur.

Riassunto in italiano

Linguaggi di ordine superiore: dualità e tecniche di bisimulazione

Il comportamento di processi concorrenti può essere rappresentato attraverso *calcoli di processi*, che sono semplici linguaggi formali attraverso i quali è possibile dimostrare specifici risultati matematici su interazioni tra processi (Capitolo 1).

Un semplice ma espressivo calcolo di processi è, ad esempio, CCS (Calculus of Communicating Systems). Un altro esempio comune è il pi-calcolo, la cui espressività è maggiore grazie ad un meccanismo di comunicazione tramite canali. Con il pi-calcolo è inoltre possibile utilizzare sistemi di tipi, che sono in grado di catturare il comportamento di processi che evolvono in ambienti con maggiori vincoli. Il pi-calcolo permette inoltre di codificare il lambda calcolo, che a sua volta cattura l'essenza delle computazioni sequenziali (Capitolo 2).

Sono stati proposti diversi calcoli di processi, con livelli differenti di complessità. Una proprietà di alcuni di questi calcoli è nota come "simmetria": calcoli come CCS, il pi-calcolo con mobilità interna o calcoli "con fusioni" sono invarianti rispetto a un'operazione di dualità tra invio e ricezione (Sezione 3.1 e 3.2). Il primo contributo di questa tesi consiste nell'utilizzo di questa dualità nel pi-calcolo (Capitolo 3). Il pi-calcolo non è stabile rispetto alla dualità, quindi costruiamo per saturazione una estensione conservativa del pi-calcolo tipato che risulta stabile. Questa estensione permette di stabilire una corrispondenza tra due codifiche del lambda-calcolo nel pi-calcolo, che appaiono molto diverse tra loro (Sezione 3.3).

La corrispondenza viene stabilita attraverso la costruzione di un nuovo calcolo simmetrico tipato. Questo porta dunque a indagare la possibilità di avere sistemi di tipi per i calcoli simmetrici esistenti. Il resto del capitolo si concentra su questo tema e si conclude con una risposta negativa: viene stabilito un teorema di impossibilità, che afferma che i calcoli con fusioni non possono supportare l'aggiunta di sistemi di tipi, salvo casi banali (Sezione 3.4).

Attraverso la dimostrazione di questo teorema, scopriamo un vincolo fondamentale di questi calcoli che impedisce l'utilizzo di tipi, ossia il fatto che induce una relazione di equivalenza tra i canali di comunicazione. Questa relazione è in particolare evidente nel calcolo con "fusioni esplicite". Rilassando questo vincolo e permettendo ai canali di comunicazione di essere invece in una relazione di preordine si ottiene il "pi-calcolo con preordini", un nuovo calcolo nello stile dei calcoli con fusioni, di cui vengono in seguito studiate le proprietà (Sezione 4).

Come nel calcolo con fusioni esplicite, un operatore del calcolo chiamato "arco" rappresenta la relazione indotta sui nomi, associando i nomi che sono nella relazione globale di preordine. Questi archi agiscono attraverso un semplice sistema di riscrittura, e definiscono il comportamento del calcolo (Sezione 4.1). Si può inoltre vedere come il pi-calcolo con preordini supporti in maniera diretta un sistema di tipi (Sezione 4.2).

La semantica con riscrittura è "avida", in quanto gli archi possono reagire con qualsiasi prefisso in qualsiasi momento. Presentiamo una semantica più "pigra", che utilizza gli archi esclusivamente al momento della comunicazione (Sezione 4.3). Questa semantica descrive una strategia di riduzione più tradizionale, che permette di avere proprietà più interessanti e maggiori corrispondenze con i calcoli esistenti.

Il calcolo è inoltre in grado di gestire i nomi privati, che si trovano nella versione originale del pi-calcolo ma non nei calcoli con fusioni (Sezione 4.4). Questa capacità di creare nuovi nomi privati è ampiamente utilizzata nelle implementazioni, fatto che si aggiunge alla possibilità di avere un calcolo con un sistema di tipi.

Illustriamo in seguito la semantica comportamentale del pi-calcolo "pigro" con preordini e le sue specificità, attraverso lo sviluppo di un sistema di transizione etichettate (LTS), una bisimulazione e dimostrazioni tecniche di risultati di congruenza, alcuni dei quali sono formalizzati in Coq (Sezione 4.5). Una seconda caratterizzazione è data sotto forma di un insieme di leggi comportamentali che catturano completamente la semantica. Questa assiomatizzazione può essere adattata per ottenere, per la prima volta, una assiomatizzazione del calcolo con fusioni esplicite (Sezione 4.6).

Il pi-calcolo con preordini è sufficientemente espressivo da codificare il pi-calcolo e i calcoli con fusioni, con una corrispondenza particolarmente fine per i processi asincroni (Sezione 4.7). Infine, notiamo che la

codifica del pi-calcolo induce una naturale restrizione sintattica in calcoli con fusioni e preordini. Si dimostra che i calcoli coincidono sul dominio di tale restrizione e che è inoltre possibile aggiungervi un sistema di tipi, scoprendo così un ulteriore modo per recuperare l'uso dei tipi nei calcoli con fusioni (Sezione 4.8).

Ci occupiamo in seguito (Capitolo 5) di metodi di dimostrazione per la semantica comportamentale di processi, con particolare attenzione al metodo di bisimulazione. Quest'ultimo riesce a catturare l'equivalenza desiderata in diversi linguaggi di ordine superiore, come il pi-calcolo e il lambda-calcolo. Una prova di bisimulazione si articola in due parti principali: una prima parte in cui si costruisce una relazione che deve contenere gli oggetti che si vogliono dimostrare equivalenti, ed una seconda parte in cui dimostriamo che questa relazione è una bisimulazione. La seconda parte della dimostrazione richiede solitamente di espandere la relazione, fatto che a sua volta implica l'iterazione alternata delle due parti che costituiscono il metodo di bisimulazione, fino al raggiungimento di una relazione che soddisfa le clausole di bisimulazione. Parte di questa iterazione è ridondante, e può infatti essere evitata o semplificata drasticamente, a volte in modo decisivo per completare la prova, utilizzando tecniche di bisimulazioni "up to" (ad esempio, una bisimulazione "up to context" può essere molto più piccola di una bisimulazione ordinaria). Pous e Sangiorgi hanno sviluppato una teoria generale delle bisimulazioni up to che fornisce un quadro generale e modulare in cui utilizzare queste tecniche: quando due tecniche sono dimostrate essere corrette, la loro combinazione è automaticamente derivata (Sezione 5.1).

Tuttavia questa teoria si applica solamente nel caso in cui i sistemi esaminati siano sistemi "semplici", come automi o CCS. Tecniche di bisimulazione per linguaggi di ordine superiore (come il pi-calcolo, il lambda-calcolo e le loro varianti) devono purtroppo essere adattate di volta in volta a partire dalle prove della teoria generale, e hanno il grave inconveniente di richiedere nuove prove per ciascuna combinazione di tecniche, contrariamente a quanto avviene nella teoria generale.

Questa tesi contribuisce all'applicazione della teoria generale delle tecniche up to ai linguaggi di ordine superiore. L'approccio adottato consiste nello stabilire una corrispondenza precisa tra un linguaggio di ordine superiore e un sistema di transizione del primo ordine, affinché la bisimulazione indotta su quest'ultimo coincida con l'equivalenza comportamentale desiderata sul linguaggio di ordine superiore. Applichiamo questo metodo al pi-calcolo, utilizzando processi con nomi (Sezione 5.2), poi al lambda-calcolo, sulla base di una nozione specifica di bisimulazione basata sul concetto di "ambiente" (Sezione 5.3). Mostriamo infine come trattare un linguaggio con primitive di programmazione imperativa (Sezione 5.4) e altri linguaggi, come il pi-calcolo di ordine superiore (Sezione 5.5).

In ciascuno di questi casi, grazie alla descrizione al primo ordine dei linguaggi, ereditiamo tutte le tecniche della teoria generale. Dimostriamo inoltre la correttezza delle tecniche "up to context", che sono specifiche per ciascuno di questi calcoli e formano una solida base per le dimostrazioni di bisimulazione. Le condizioni di correttezza di queste tecniche non sono sempre rispettate, e possono dipendere da come abbiamo costruito i sistemi di transizione del primo ordine. Questo studio si propone di guidare il lettore nella costruzione di sistemi di transizione che ammettono queste tecniche, al fine di gestire altri linguaggi di ordine superiore.

Contents

Chapter 1

Introduction

1.1 Process algebras

Concurrency theory models interacting systems as mathematical objects, making possible formal reasoning about complex distributed systems and parallel computing. The notion of *process algebra* formalises those systems: the base objects of these algebras are the *processes* (P, Q, ...). They can be defined and composed using operators, and can be related through laws (written $P \equiv Q$).

The key operator of process algebras is parallel composition. If P and Q are two processes, then their parallel composition, written $P \mid Q$, is the process that behaves as if P and Q were executed simultaneously, authorising communications between P and Q. This operator has a neutral element, written 0, which is the process that does nothing. Because of this, composing 0 with another process P will result in a process that behaves exactly like P, hence we require in our algebra that $P \mid 0 \equiv P$. In fact, parallel composition must form a commutative monoid: $P \mid Q \equiv Q \mid P$ and $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$.

Now that parallel composition allows the actual distribution of processes, communication must be accounted for. To that end, processes are provided with communication channels, on which they can send and receive data. Sending some data d on channel a is written $\overline{a}\langle d\rangle.P$ where P is the continuation process. Sending is a blocking action: P will only be run once d has been received. Receiving some unknown data x on channel a is written $a(x).Q$, process Q being executed only once all data has been received and stored in x. Composing these two processes will trigger an interaction through channel a:

$$\overline{a}\langle d\rangle.P \mid a(x).Q \quad \longrightarrow \quad P \mid Q\{d/x\} \tag{1.1}$$

where d has indeed replaced variable x in Q after the communication (which we write $Q\{d/x\}$). This reduction is usually the only reduction of the calculus, and is enough to represent parallel computing.

Another useful primitive is the hiding of communication channels: suppose that a is a channel. Then, in the process $(\boldsymbol{\nu} a)P$, the communication channel a can only be used inside of process P. Process $(\boldsymbol{\nu} a)P$ can also be viewed as a process that creates for itself a new private channel a.

We have seen five primitives for now, summed up in the following BNF-style grammar:

$$P \;::=\; 0 \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \overline{a}\langle d\rangle.P \;\mid\; a(x).Q \;. \tag{1.2}$$

We did not specify how x can be used inside Q, or even what sort of objects x and d are. There are many ways to answer these questions, resulting in different process calculi:

- the simplest way is to forbid any usage of x in Q; then both x and data d are useless, hence we can write $\overline{a}.P$ instead of $\overline{a}\langle d\rangle.P$ and $a.Q$ instead of $a(x).Q$. The resulting calculus is CCS (Section 2.1) which is surprisingly expressive despite its simple semantics.
- The next step is to say that x and d are channels: then communication channel names (or simply "names") can be sent across the network of processes. Process calculi with this ability are called

name-passing calculi, they generally have a more complicated semantics but are very expressive. The π-calculus (Section 2.2) is the archetypal name-passing calculus.

In some presentations, a and d are names, but x is a *variable* waiting to be instantiated, which is a different object. We choose here to invariably call them names, as this will make for a more uniform presentation of the other calculi we study.

- Finally x and d can also stand for processes: a process sends the code of another process and the receiver has then the ability to run it. We call those calculi *higher-order*; a famous example is the higher-order pi-calculus, studied in Section 5.5.2.

We do not consider here the many other sorts of data that can be found in the literature, like constants, functions, or compound patterns. The more sorts of data one can communicate, the more expressive the calculus is. But there are counter-intuitive quirks: for example, the ability to communicate plain processes only is significantly less expressive than name-passing alone. Indeed, the former is surprisingly close to plain CCS, while name-passing can encode process communication fairly easily. In the first parts of this thesis, we will focus on name-passing calculi.

1.2 Bisimulations

Concurrent processes are generally given meaning through a notion of behavioural equivalence: one says that P and Q are *behaviourally equivalent* ($P \simeq Q$) if they behave the same way in all environments. In all process calculi we consider in this thesis, reduction-closed barbed congruence will be our reference behavioural equivalence. Its definition (Definition 2.1.4) is based on the reduction relation $\longrightarrow$ of the calculus, on the closure by algebraic laws $\equiv$ and quantifies over a set of environments that is generally infinite. Because of this, a proof of $P \simeq Q$ has to include an infinite relation on processes, on which it is difficult to form case analyses.

The usual solution to this problem is usually to *generalise* the reduction relation $\longrightarrow$ to make it compositional. For example, the reduction from a composition of processes $P_1 \mid P_2 \longrightarrow P'$ can have several origins: either a reduction from P_1, or a reduction from P_2, or a composition of two actions: one from P_1, and another from P_2. This idea allows the decomposition of a reduction relation into a Labelled Transition System (LTS), a compositional notion of interaction, easier to manipulate. Especially, processes need not be considered "up to" the algebraic laws $\equiv$ seen in Section 1.1. For example, the reduction in (1.1) can be decomposed into the following two actions:

$$\overline{a}\langle d\rangle.P \xrightarrow{\overline{a}\langle d\rangle} P \qquad\qquad a(x).Q \xrightarrow{ad} Q\{d/x\}$$

that can be composed back together using a communication rule:

$$\frac{\overline{a}\langle d\rangle.P \xrightarrow{\overline{a}\langle d\rangle} P \qquad a(x).Q \xrightarrow{ad} Q\{d/x\}}{\overline{a}\langle d\rangle.P \mid a(x).Q \xrightarrow{\tau} P \mid Q\{d/x\}} \ ,$$

where τ is the label of a silent action, or internal action: transition $\xrightarrow{\tau}$ is essentially the same relation as reduction $\longrightarrow$.

Given some LTS—which is the compositional version of a reduction relation—it is possible to establish the notion of (labelled) bisimulation—which is the compositional version of the reduction-closed barbed congruence: a relation $\mathcal{R}$ on processes is a bisimulation if, when it relates two processes ($P \mathcal{R} Q$) and one of them does an action ($P \xrightarrow{\mu} P'$ or $Q \xrightarrow{\mu} Q'$), then the other process does the same action ($Q \xrightarrow{\mu} Q'$ or $P \xrightarrow{\mu} P'$, respectively) and the two resulting processes are still related ($P' \mathcal{R} Q'$).

Graphically, $\mathcal{R}$ is a bisimulation if it validates the two diagrams in Figure 1.1 for every action μ. We use plain lines for universal quantifiers (in premises), and dotted lines for existential quantifiers (in conclusions).

Two processes are bisimilar ($P \sim Q$) if there is a bisimulation relating them. Bisimilarity $\sim$ is then the union of all bisimulations or, equivalently, the largest bisimulation. If the LTS is designed correctly, then

Figure 1.1: Bisimulation diagrams

$P \sim Q$ if and only if $P \simeq Q$, providing an efficient proof technique for equivalence of processes. This proof technique can be further improved using up-to techniques, on which we focus in Chapter 5.

1.3 Expressiveness and symmetry

The design of programming languages and process calculi gravitates around several principles, among which the notion of minimality is important for foundational languages. Minimality expresses that no base construct can be encoded into a combination of others. One example is the internal choice operator $\oplus$ (sometimes called nondeterministic choice operator in sequential calculi), defined as follows:

$$\frac{}{P \oplus Q \longrightarrow P} \qquad \frac{}{P \oplus Q \longrightarrow Q} \ . \tag{1.3}$$

If we assume the presence of the operators seen in Section 1.1, then internal choice can be encoded as follows, where a is a fresh name:

$$[\![P \oplus Q]\!] \triangleq (\boldsymbol{\nu} a)(\overline{a} \mid a.P \mid a.Q) \ .$$

The only actions of the above encoding are exactly the reductions in (1.3). This means that process calculi with operators $\boldsymbol{\nu}$, $\overline{a}$, $a.$, and $\mid$ are inherently non-deterministic, and is the reason why operator $\oplus$ is not included in foundational process calculi. Consider now instead the external choice operator $+$, which can be defined as follows, not in terms of reductions but in terms of transitions (cf. discussion in previous section):

$$\frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'} \qquad \frac{Q \xrightarrow{\mu} Q'}{P + Q \xrightarrow{\mu} Q'} \tag{1.4}$$

While they could be seen as having roughly similar behaviours, $+$ cannot be encoded as simply as $\oplus$ can [Pal97, Nes97]. This is why the external choice operator has several variants and can be said to have significant expressiveness.

Instead of encoding a lone construct into other more basic constructs, one can encode a whole variant of a calculus into another. For example, the polyadic π-calculus [Mil92] (where emitted data are tuples of names) can be encoded into the monadic π-calculus (sending only one name at a time) under some correspondence conditions.

Knowing a variant of a calculus can be encoded into the original calculus provides robustness. For example, to encode call-by-name λ-calculus into the π-calculus (Section 2.4, [Mil90]), one can switch from monadic to polyadic prefixes for simplicity, without belittling the result about relative expressive powers of λ and π.

Another desirable property of a calculus is that each operator has only one role: even if this is defined informally, this makes the system easier to analyse. For example in CCS, restriction and parallel composition have the distinct roles of binding and distribution, even if those can interact. On the contrary in the π-calculus, the input construct is also a binder, blurring the differences between constructs. This motivates some sort of separation between the binding and input constructs, suggesting a symmetric syntax with respect to input/output polarities (since output is not binding). CCS has this separation and symmetry

properties, but it does not have the expressive power of the π-calculus. The expressiveness gap comes from the name-passing ability of the π-calculus.

There are however calculi that reach an expressiveness of the level of the π-calculus, using different implementations of the name-passing ability, and that have the same symmetry property as CCS. The fusion calculi (Section 3.2, [PV98a, GW00]) and the π-calculus with internal mobility (Section 3.1, [San96b]) are such examples. In addition, the fusion calculi have only one binder, so they have a property of separation of roles between constructs, as in CCS. Finally, those calculi generate a simpler and clearer behavioural theory ([PV98a, WG04, San96b]). Rather informally speaking, it appears that symmetry and simplicity go hand in hand: simplicity calls for minimality, hence the symmetry of constructs, and symmetry yields simple behavioural theories.

Contributions

We use symmetry as a tool to establish behavioural properties in the π-calculus, using this to relate existing encodings of the λ-calculus in the π-calculus (Section 3.3). We prove that fusion calculi are incompatible with subtyping and capability types in channel-based type systems and remark that such type systems are irrelevant for the π-calculus with internal mobility (Section 3.4).

We build a process calculus (written the $\pi\mathrm{P}$-calculus) that lifts this impossibility, thanks to a notion of preorder on names (Chapter 4). Our calculus has only one binder and have a channel-based type system with subtyping and capabilities, and is not captured by existing calculi. In addition, our calculus gets back a notion of privacy of names that was featured by the π-calculus but not by fusion calculi. We then characterise the behavioural theory of this calculus in two ways: by a compositional bisimulation semantics and by establishing an axiomatisation of the calculus. It turns our the latter can be adapted to build an axiomatisation of the *explicit* fusion calculus, for which no axiomatisation have been established before.

In a second part we extend the scope of application of bisimulations. The theory of up-to techniques provides an impressive number of tools to establish bisimulation results, as well as a way to incorporate new tools and to combine them together. Unfortunately this theory only applies to a restricted number of systems, those who have a "first-order" representation, for example CCS and finite automata. For other systems that do not have such representations (notably, known bisimulations for π-calculi and λ-calculi), one needs to adapt each up-to technique along with its proof of correctness, which is usually redundant with other proofs. Our contribution (Chapter 5) is to find first-order representations for some of those calculi so that they inherit all the already-developed theory of up-to techniques. We also provide and prove correct the tools specific to each representation, i.e. the up to context techniques, which gives shorter and clearer proofs of correctness and congruence. We apply this method for the π-calculus, the λ-calculus, the λ-calculus with references, and the higher-order π-calculus, and try to provide guidelines and the problems to be expected in future applications of this method.

Outline

- Chapter 1 is this introduction.
- Chapter 2 gives some background, containing definitions common to most process calculi mentioned in this work: CSS, the π-calculus, their bisimulations, the typing discipline of π, and the λ-calculus.
- Chapter 3 studies symmetry in process calculi: we list the existing symmetric calculi (Sections 3.1 and 3.2), we explain how to use symmetry to establish behavioural properties (Section 3.3), and the incompatibility of types with symmetric calculi (Section 3.4).
- Chapter 4 describes the $\pi\mathrm{P}$-calculus.
- Chapter 5 gives first-order representations of calculi, and their corresponding techniques.

Published works

Section 3.3, about using symmetry as a tool to relate processes, has been published in CONCUR [HMS12]. Section 3.4 providing the incompatibility of types, Section 4.1, and several parts of Chapter 4 introducing $\pi\mathrm{P}$, have been published together in LICS [HMS13]. The complete behavioural theory of $\pi\mathrm{P}$ in Sections 4.5 and 4.6 is to appear in the proceedings of FSEN [HMX15]. Most of Chapter 5 on applications of the theory of up-to techniques appears in CONCUR [MPS14].

Chapter 2

Background

2.1 The Calculus of Communicating Systems

2.1.1 Processes and reductions

The Calculus of Communicating Systems (CCS) is the most basic process calculus based on the operators introduced in Section 1.1. We first describe it by giving the following, simplified grammar of CCS, which is a special case of Grammar (1.2) when no data is transmitted. We give further below the details of the operators of choice and of replication, which are usually presented together with CCS.

$$P \ ::= \ 0 \ \mid \ P \mid Q \ \mid \ (\boldsymbol{\nu} a)P \ \mid \ \overline{a}.P \ \mid \ a.P \ .$$

Above, a ranges over a countable set of names $\mathcal{N}$, the names being the channels of the calculus. They can be bound by the restriction operator $\boldsymbol{\nu}$. To define this formally, we define the set $\mathrm{fn}(P)$ of *free names* of P as follows:

$$\mathrm{fn}(0) \triangleq \emptyset \qquad \mathrm{fn}(P \mid Q) \triangleq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$$

$$\mathrm{fn}((\boldsymbol{\nu} a)P) \triangleq \mathrm{fn}(P) \setminus \{a\} \qquad \mathrm{fn}(\overline{a}.P) \triangleq \mathrm{fn}(a.P) \triangleq \{a\} \cup \mathrm{fn}(P) \ .$$

We present here two ways to give meaning to CCS processes: the first is the notion of *reduction* of processes, and the second is the notion of *transitions* between processes. As explained in Section 1.2 the latter is more compositional, hence easier to reason about, while the former may be easier to understand. We give first the definition of reductions, for which we first need to define structural congruence.

Definition 2.1.1 (Structural congruence)**.** Structural congruence is the least congruence relation $\equiv$ satisfying the following laws:

$$P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \equiv Q \ \text{ if } P =_\alpha Q$$

$$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P \qquad (\boldsymbol{\nu} x)0 \equiv 0 \qquad (\boldsymbol{\nu} x)(P \mid Q) \equiv (\boldsymbol{\nu} x)P \mid Q \ \text{ if } a \text{ is not a free name of } Q.$$

where $=_\alpha$ is alpha conversion, i.e. in this case $(\boldsymbol{\nu} a)P =_\alpha (\boldsymbol{\nu} b)P\{b/a\}$ when $b \notin \mathrm{fn}(P)$ where $\{b/a\}$ is the capture-avoiding substitution of a with b.

Definition 2.1.2 (Reduction)**.** The reduction relation of CCS, written $\longrightarrow$, is the least relation generated by the following rules:

$$\frac{}{\overline{a}.P \mid a.Q \longrightarrow P \mid Q} \qquad \frac{P \longrightarrow P'}{P \mid R \longrightarrow P' \mid R} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} a)P \longrightarrow (\boldsymbol{\nu} a)P'} \qquad \frac{P \equiv \longrightarrow \equiv P'}{P \longrightarrow P'}$$

where $\equiv\longrightarrow\equiv$ stands for the relational composition of $\equiv$, $\longrightarrow$, and $\equiv$. When not clear from the context, we specify in subscript of $\longrightarrow$ the calculus we are working with, and so we write $\longrightarrow_{\mathrm{CCS}}$ for reduction in CCS.

Remark how the inactive process 0 cannot perform any action or interaction and is only used as the first building block of processes. In the following, we will often omit it when it is a continuation of a prefix. For example, we write $\overline{a} \mid a.(a \mid b)$ instead of $\overline{a}.0 \mid a.(a.0 \mid b.0)$.

2.1.2 Barbed congruence

Now that we have a notion of reduction $\longrightarrow_{\mathrm{CCS}}$, we are a few steps away from getting the induced equivalence, but first we need a notion of observable. Intuitively, a barb (or observability predicate) at a holds for a process when the environment can detect that the process can accept an offer of interaction on channel a. More precisely:

Definition 2.1.3 (Barbs)**.** Process P has the barb $\overline{a}$, written $P \downarrow_{\overline{a}}^{\mathrm{CCS}}$, if for a fresh name w, its composition with the tester process $a.\overline{w}$ reduces in one step to a process in which $\overline{w}$ appears not guarded by a prefix. More precisely:

$$P \downarrow_{\overline{a}}^{\mathrm{CCS}} \quad \text{iff} \quad P \mid a.\overline{w} \longrightarrow_{\mathrm{CCS}} R \mid \overline{w} \quad \text{for some process } R \text{ and } w \notin \mathrm{fn}(P) \ .$$

Similarly, we define $P \downarrow_{a}^{\mathrm{CCS}}$ with the tester $\overline{a}.\overline{w}$:

$$P \downarrow_{a}^{\mathrm{CCS}} \quad \text{iff} \quad P \mid \overline{a}.\overline{w} \longrightarrow_{\mathrm{CCS}} R \mid \overline{w} \quad \text{for some process } R \text{ and } w \notin \mathrm{fn}(P) \ .$$

Traditionally, barbs are defined syntactically, directly in terms of structural congruence. Instead, we use tester processes $a.\overline{w}$ and $\overline{a}.\overline{w}$. This way, the definition still makes sense for other calculi in the remainder of this work. (Note that for asynchronous calculi, the tester $\overline{a}.\overline{w}$ would not be legitimate syntax, and indeed there is no input barbs in asynchronous equivalences—see Section 5.5.1) This definition of barbs is inspired from the notion of success in testing equivalence [DH84], the $\overline{w}$ emulating the special signal usually written ω. Even if our notion of barb coincides with the usual one, it could be slightly different: the equivalence from Definition 2.1.4 is robust enough with respect to most choices of observability predicates.

Barbed congruence for any calculus In fact, we can define barbed congruence for an arbitrary calculus $\mathcal{L}$. The definition only requires a reduction relation, $\longrightarrow_{\mathcal{L}}$, and a notion of barb on names, $\downarrow_a^{\mathcal{L}}$. For CCS we could take it to be the output barbs, but it does not matter much: any kind of nontrivial observable would do. Since we define a congruence, we also assume a notion of contexts to compose processes: if P is a process in $\mathcal{L}$ and if C is a context of $\mathcal{L}$ (i.e. a process with one hole $[\cdot]$), then we write $C[P]$ for the process obtained by replacing the hole $[\cdot]$ with P in C.

Definition 2.1.4 (Reduction-closed barbed congruence)**.** Let $\mathcal{L}$ be a process calculus, in which a reduction relation $\longrightarrow_{\mathcal{L}}$ and barb predicates $\downarrow_a^{\mathcal{L}}$, for each a in a given set of names, have been defined. A relation $\mathcal{R}$ on the processes of $\mathcal{L}$ is:

- *context-closed* if $P \mathcal{R} Q$ implies $C[P] \mathcal{R} C[Q]$ for each context C of $\mathcal{L}$;
- *barb-preserving* if $P \mathcal{R} Q$ implies that for any name a, $P \downarrow_a^{\mathcal{L}}$ if and only if $Q \downarrow_a^{\mathcal{L}}$;
- *reduction-closed* if, whenever $P \mathcal{R} Q$:
  - $P \longrightarrow_{\mathcal{L}} P'$ implies that for some Q', $Q \longrightarrow_{\mathcal{L}} Q'$ and $P' \mathcal{R} Q'$,
  - $Q \longrightarrow_{\mathcal{L}} Q'$ implies that for some P', $P \longrightarrow_{\mathcal{L}} P'$ and $P' \mathcal{R} Q'$.

Then *reduction-closed barbed congruence in* $\mathcal{L}$, written $\simeq_{\mathcal{L}}$, is the largest relation on the processes of $\mathcal{L}$ that is context-closed, reduction-closed, and barb-preserving.

2.1.3 Labelled transition system, bisimulation

Section 1.2 explains how transitions can be seen as a compositional presentation of reductions. This also means that they can be used to describe each operator of the calculus independently. The labels of the transitions are ranged over by μ, and can be: *input actions* a, *output actions* $\overline{a}$, or *silent actions* τ where a ranges over names and τ is a fixed symbol:

$$\mu \ ::= \ a \ \mid \ \overline{a} \ \mid \ \tau \ .$$

Label τ is an internal computation step of a process, we call it *invisible*. If label μ is not τ, we call it a *visible* label. We list below the operators of the calculus, giving for each an informal account of its behaviour and the formal rules giving the possible transitions or "actions" that one can derive from it:

- *input*: $a.P$ can do an action a and then proceeds to become P:

$$\frac{}{a.P \xrightarrow{a} P} \tag{2.1}$$

- *output*: symmetrically, $\overline{a}.P$ does the output action $\overline{a}$ to P:

$$\frac{}{\overline{a}.P \xrightarrow{\overline{a}} P} \tag{2.2}$$

- *restriction*: the hiding operator indeed hides the name: if P can do action μ then $(\boldsymbol{\nu} a)P$ can do μ as well, as long as μ does not mention a:

$$\frac{P \xrightarrow{\mu} P' \qquad a \notin \mathrm{n}(\mu)}{(\boldsymbol{\nu} a)P \xrightarrow{\mu} (\boldsymbol{\nu} a)P'} \tag{2.3}$$

  where $\mathrm{n}(\mu)$ is the set of names mentioned by μ and is defined as $\mathrm{n}(\overline{a}) \triangleq \mathrm{n}(a) \triangleq \{a\}$ and $\mathrm{n}(\tau) \triangleq \emptyset$

- *parallel composition* has two roles: it lets transitions of either side go through and allows communication between the two sides. Hence, either one of the components does an action, the other being left unaffected:

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{Q \xrightarrow{\mu} Q'}{P \mid Q \xrightarrow{\mu} P \mid Q'} \ , \tag{2.4}$$

  or the two components synchronise: if the first component does an input action a and the second component does the corresponding output action $\overline{a}$ on the same name (or vice versa), then the whole parallel composition does a τ action:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{\overline{a}} P' \qquad Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \tag{2.5}$$

- *nil*: the process 0 cannot interact with anything, and hence has no transition.

Note how inputs and outputs are treated symmetrically. From a meta-theoretic point of view, this is enough to say that we can exchange input and outputs without changing the behaviour of processes. We give a precise meaning to this in Proposition 3.0.3 (Chapter 3).

Putting together rules (2.1), (2.2), (2.3), (2.4), (2.5) yields for each label μ a relation $\xrightarrow{\mu}$, defining a labelled transition system (LTS) for CCS processes. Given such an LTS, the induced bisimulation is defined as follows:

Definition 2.1.5 (Bisimulation)**.** A relation $\mathcal{R}$ is a *bisimulation* if and only if, whenever $P \mathcal{R} Q$:

- if $P \xrightarrow{\mu} P'$ then, for some Q', $Q \xrightarrow{\mu} Q'$ and $P' \mathcal{R} Q'$, and
- if $Q \xrightarrow{\mu} Q'$ then, for some P', $P \xrightarrow{\mu} P'$ and $P' \mathcal{R} Q'$.

Bisimilarity $\sim$ is the union of all bisimulations. We say that P and Q are *bisimilar* if $P \sim Q$.

Graphically, $\mathcal{R}$ is a bisimulation if it validates the bisimulation diagrams from Figure 1.1 (Section 1.2), and bisimilarity is, equivalently, the largest relation that validates them. The property of interest is that bisimilarity coincides with reduction-closed barbed congruence:

Theorem 2.1.6. *For all CCS processes P and Q, $P \simeq Q$ if and only if $P \sim Q$.*

To show this correspondence, there are two directions. The first is to show that bisimilarity is a congruence, which can be done in several ways. One can prove congruence manually, by induction on the context—the most interesting case is the one for parallel composition, i.e. $P \sim Q$ implies $P \mid R \sim Q \mid R$. But there are different ways to be more modular and independent of the syntax of the calculus. For instance:

- Milner and Sangiorgi [MS92] invoke the fact that the transitions rules for CCS are well-founded and in *tyft/tyxt* format which implies that $\sim$ is a congruence [GV89].
- Pous has developed [Pou08] the notion of *initial contexts* in order to derive proof techniques for bisimulation (seen more in depth in Section 5.1.1) by focusing on one operator at a time. The congruence of $\sim$ is a pleasant side effect of the correctness of those techniques.

The second direction is a quite simple case analysis on visible labels: $P \xrightarrow{\overline{a}} \equiv P'$ if and only if, for some fresh w, there exists a P_1 such that $P \mid a.(w \mid \overline{w}) \longrightarrow P_1 \longrightarrow P'$, with $P_1 \downarrow_w^{\text{CCS}}$ and $P' \not\downarrow_w^{\text{CCS}}$, and symmetrically for input.

2.1.4 Weak behavioural theory

Internal mechanisms of agents are sometimes ignored to allow more general reasoning or to relate an implementation to its specification. The notion of weak behavioural theory is designed to ignore such internal transitions. In the following, we define weak versions of relations and predicates we defined before.

Weak reduction $\Longrightarrow_{\mathcal{L}}$ is the reflexive transitive closure of the internal reduction $\longrightarrow_{\mathcal{L}}$. The weak observability predicate $\Downarrow_a^{\mathcal{L}}$ is defined by composing a weak reduction and a strong barb:

$$\Longrightarrow_{\mathcal{L}} \triangleq \longrightarrow_{\mathcal{L}}^* \qquad\qquad P \Downarrow_a^{\mathcal{L}} \iff P \Longrightarrow_{\mathcal{L}} \downarrow_a^{\mathcal{L}} \quad .$$

Weak reduction-closed barbed congruence is defined as the strong version, replacing the strong observability predicate and reduction with their weak variants. In fact, we can make the definition a little less demanding and only ask for the answers to the challenges to be weak:

Definition 2.1.7 (Weak reduction-closed barbed congruence)**.** A relation $\mathcal{R}$ on the processes of $\mathcal{L}$ is:

- *weakly barb-preserving* if $P \mathcal{R} Q$ implies that for any name a, $P \downarrow_a^{\mathcal{L}}$ implies $Q \Downarrow_a^{\mathcal{L}}$;
- *weakly reduction-closed* if $P \mathcal{R} Q$ and $P \longrightarrow_{\mathcal{L}} P'$ imply there is Q' such that $Q \Longrightarrow_{\mathcal{L}} Q'$ and $P' \mathcal{R} Q'$.

Then *weak reduction-closed barbed congruence in $\mathcal{L}$*, written $\cong_{\mathcal{L}}$, is the largest symmetric relation on the processes of $\mathcal{L}$ that is context-closed, weakly reduction-closed, and weakly barb-preserving.

To capture weak reduction-closed barbed congruence, one can build weak bisimilarity, defined through weak transitions.

Since the following is independent of the syntax of the calculus we consider, we get rid of the indexing with $\mathcal{L}$—keeping the special transition τ. The formal framework for this is defined in the beginning of Section 5.1.

Definition 2.1.8 (Weak transitions)**.** Relation $\Longrightarrow$, and indexed relations $\stackrel{\hat{\cdot}}{\longrightarrow}$, $\stackrel{\hat{\cdot}}{\Longrightarrow}$ and $\stackrel{\cdot}{\Longrightarrow}$ are defined as follows:

- $\Longrightarrow \triangleq \stackrel{\tau}{\longrightarrow}^*$
  (where $\mathcal{R}^*$ ($P_0 \; \mathcal{R}^* \; P_n$ iff $n \geq 0$ and $P_0 \; \mathcal{R} \; P_1 \; \mathcal{R} \; \ldots \; \mathcal{R} \; P_n$) is the reflexive transitive closure of $\mathcal{R}$)
- $\stackrel{\hat{\mu}}{\longrightarrow} \triangleq \stackrel{\mu}{\longrightarrow}$ if $\mu \neq \tau$ and $\stackrel{\hat{\mu}}{\longrightarrow} \triangleq \longrightarrow^=$ if $\mu = \tau$
  (where $\mathcal{R}^=$ ($P \; \mathcal{R}^= \; Q$ iff $P = Q$ or $P \; \mathcal{R} \; Q$) is the reflexive closure of $\mathcal{R}$)
- $\stackrel{\hat{\mu}}{\Longrightarrow} \triangleq \Longrightarrow\stackrel{\hat{\mu}}{\longrightarrow}\Longrightarrow$
- $\stackrel{\mu}{\Longrightarrow} \triangleq \Longrightarrow\stackrel{\mu}{\longrightarrow}\Longrightarrow$

Definition 2.1.9 (Weak bisimulation)**.** A relation $\mathcal{R}$ is a *weak bisimulation* if and only if, whenever $P \; \mathcal{R} \; Q$:

- if $P \stackrel{\mu}{\longrightarrow} P'$ then, for some Q', $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $P' \; \mathcal{R} \; Q'$, and
- if $Q \stackrel{\mu}{\longrightarrow} Q'$ then, for some P', $P \stackrel{\hat{\mu}}{\Longrightarrow} P'$ and $P' \; \mathcal{R} \; Q'$.

Weak bisimilarity $\approx$ is the union of all weak bisimulations, and P and Q are *weakly bisimilar* if $P \approx Q$.

On CCS processes as defined in Section 2.1.1 (without the choice operator), weak bisimilarity coincides with weak reduction-closed barbed congruence:

Theorem 2.1.10. *For all CCS processes P and Q, $P \approx Q$ if and only if $P \approx Q$.*

2.1.5 Common extensions

We define in this section two common operators of CCS that are useful in our setting: choice and replication.

Replication (!) The replication $!P$ of a process P represents an arbitrary number of copies of P in parallel. The grammar is extended as follows:

$$P \;\; ::= \;\; \ldots \;\; \mid \;\; !P \;\; .$$

It can be defined both in terms of reduction by adding one law to structural congruence, and in terms of transitions, by adding one rule:

$$!P \equiv !P \mid P \qquad\qquad \frac{!P \mid P \stackrel{\mu}{\longrightarrow} P'}{!P \stackrel{\mu}{\longrightarrow} P'} \tag{2.6}$$

Both mean that $!P$ stands in principle for an infinite parallel composition of copies of P. Theorems 2.1.6 and 2.1.10 above hold in presence of replication.

Choice (+) The choice between two processes, or their sum, represents a process that can behave as one of the two processes, possibly by reacting to the actions of the environment (contrary to internal choice, defined in (1.3), which performs this choice by itself). The grammar is extended as follows:

$$P \;\; ::= \;\; \ldots \;\; \mid \;\; P + Q \;\; .$$

Choice can be defined in terms of transitions using the two rules of (1.4), that we write again here:

$$\frac{P \stackrel{\mu}{\longrightarrow} P'}{P + Q \stackrel{\mu}{\longrightarrow} P'} \qquad\qquad \frac{Q \stackrel{\mu}{\longrightarrow} Q'}{P + Q \stackrel{\mu}{\longrightarrow} Q'} \tag{2.7}$$

With this addition, Theorem 2.1.6 holds but Theorem 2.1.10 fails. Indeed, in the weak case and in presence of arbitrary choice, reduction-closed barbed congruence is not a reasonable equivalence: context closure in presence of choice is too strong a requirement in addition to being reduction-closed, and indeed, we reach a contradiction from the expected law $a.\tau.b \cong a.b$:

$$\begin{array}{ccc} a.\tau.b & \cong & a.b \\ \overline{a} \mid a.\tau.b & \cong & \overline{a} \mid a.b \\ \downarrow & & \Downarrow \\ \tau.b & \cong & b \\ c + \tau.b & \cong & c + b \\ \downarrow & & \Downarrow \\ b & \cong & c + b \end{array} \quad \begin{array}{l} \text{(by assumption)} \\ \text{(by context closure)} \\ \\ \text{(reduction from LHS, with no } \overline{a} \text{ barb)} \\ \text{(by context closure)} \\ \\ \text{(reduction from LHS, and contradiction.)} \end{array}$$

Arbitrary choice is usually handled using a notion of barbed congruence [SW01] that does not alternate between context closure and reduction closure. Another way is to modify Definition 2.1.7 and require the relations to be closed under parallel contexts instead of arbitrary contexts.

Alternatively, most use cases of choice can be recovered by the restricted form of *guarded choice*: all sums must be of the form $\pi_1.P_1 + \cdots + \pi_n.P_n$ where π_is are prefixes ($\overline{a}$, a and sometimes[1] τ in CCS). The grammar can then be written as summations ranging over a family of indices I (depending on the calculus, I is sometimes allowed to be infinite):

$$P \;::=\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \sum_{i \in I} \pi_i.P_i \qquad\qquad \pi \;::=\; \overline{a} \;\mid\; a \;\; .$$

the base rule of the reduction relation can then be defined as follows:

$$\frac{\pi_i = \overline{a} \qquad \rho_j = a}{\sum_{i \in I} \pi_i.P_i \mid \sum_{j \in J} \rho_j.Q_j \longrightarrow P_i \mid Q_j} \tag{2.8}$$

In this work, when reduction-closed barbed congruence is used, we use only guarded choice, for which the choice of equivalence between reduction-closed barbed congruence and barbed congruence is unimportant.

Remark 2.1.11. In this section on CCS, definitions are independent from the syntax of the language. In fact, the notions of bisimulations from Definitions 2.1.5 and 2.1.9 are standard (contrarily to the ones for the π-calculus), so much that we could call standard bisimulations "CCS-like". This is the notion that will be used in Chapter 5.

Note that the addition of operators of replication (using (2.6)) and choice (using (2.7)) in terms of LTS does not depend on the calculus (provided, in the case of replication, that it already features parallel composition).

2.2 The π-calculus

The π-calculus is CCS with name-passing abilities. The data passed through channel names are names themselves, that are sometimes private names. This last bit actually provides much of the expressive power of the π-calculus. The grammar is as follows:

$$P \;::=\; 0 \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \overline{a}\langle b \rangle.P \;\mid\; a(x).P \;\; . \tag{2.9}$$

Where names are from the same set $\mathcal{N}$ and are ranged over by a, b, …, x, y, … The structural congruence $\equiv$ is defined as for CCS in Definition 2.1.1. Name x is bound in $a(x).P$, as seen in the updated definition of $\mathrm{fn}(\cdot)$ from CCS:

$$\mathrm{fn}(\overline{a}\langle b \rangle.P) \triangleq \{a, b\} \cup \mathrm{fn}(P) \qquad\qquad \mathrm{fn}(a(x).P) \triangleq \{a\} \cup (\mathrm{fn}(P) \setminus \{x\}) \;\; .$$

[1] The prefix $\tau.P$ (defined by $\tau.P \xrightarrow{\tau} P$) can be encoded using $[\![\tau.P]\!] = (\boldsymbol{\nu} u)(\overline{u} \mid u.P)$ where u does not appear in P. (The same works inside a sum.)

The reduction relation is exactly the same as in CCS, except for the base case of communication where the name is indeed passed:

Definition 2.2.1 (Reduction)**.** The reduction relation of the π-calculus is the least relation $\longrightarrow$ generated by the following rules:

$$\frac{}{\overline{a}\langle b\rangle.P \mid a(x).Q \longrightarrow P \mid Q\{b/x\}} \qquad \frac{P \longrightarrow P'}{P \mid R \longrightarrow P' \mid R} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} a)P \longrightarrow (\boldsymbol{\nu} a)P'} \qquad \frac{P \equiv \longrightarrow \equiv P'}{P \longrightarrow P'}$$

Where $Q\{b/x\}$ is the capture-avoiding substitution of b with x in Q. We sometimes write $\longrightarrow_\pi$ to avoid ambiguity.

The transitions for π need more labels: label $\overline{a}(b)$ is called *bound output*, and is useful to extend the scope of private names, which is called *scope extrusion*.

$$\mu \;::=\; ab \;\mid\; \overline{a}b \;\mid\; \overline{a}(b) \;\mid\; \tau \;.$$

We now have bound names in labels, and we write $\mathrm{bn}(\overline{a}(b)) = \{b\}$ (for other labels μ, $\mathrm{bn}(\mu) = \emptyset$). We also update the definition of $\mathrm{n}(\mu)$: $\mathrm{n}(ab) = \mathrm{n}(\overline{a}b) = \mathrm{n}(\overline{a}(b)) = \{a, b\}$ and $\mathrm{n}(\tau) = \emptyset$.

Barbs in the π-calculus are defined similarly to those of CCS:

Definition 2.2.2 (Barbs in π)**.** For a process P, output barbs and input barbs, written respectively $P \downarrow_{\overline{a}}^{\pi}$ and $P \downarrow_a^{\pi}$ are defined as follows (where $x \neq w$):

$$P \downarrow_{\overline{a}}^{\pi} \quad \text{iff} \qquad P \mid a(x).\overline{w} \longrightarrow_\pi R \mid \overline{w} \quad \text{for some process } R \text{ and } w \notin \mathrm{fn}(P)$$

$$P \downarrow_a^{\pi} \quad \text{iff} \qquad P \mid (\boldsymbol{\nu} x)\overline{a}\langle x\rangle.\overline{w} \longrightarrow_\pi R \mid \overline{w} \quad \text{for some process } R \text{ and } w \notin \mathrm{fn}(P).$$

Note that we write $\overline{w}$ as short for $(\boldsymbol{\nu} u)\overline{w}\langle u\rangle$ where $u \neq w$.

Barbed congruence for the π-calculus is defined using the generic Definition 2.1.4.

The rules defining transitions $\stackrel{\cdot}{\longrightarrow}$ (written $\stackrel{\cdot}{\longrightarrow}_\pi$ when the context is unclear) of the π-calculus in Figure 2.1 are similar to those of CCS. The differences are about extrusion: when the object b meets a restriction $(\boldsymbol{\nu} b)$, when it synchronises with an input, and some side conditions keeping the private name private. We choose the *early* flavour of the LTS [SW01], in which substitution happens in the rule for the input.

$$\frac{}{\overline{a}\langle b\rangle.P \xrightarrow{\overline{a}b} P} \qquad \frac{}{a(b).P \xrightarrow{ac} P\{c/b\}} \qquad \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} a)P \xrightarrow{\mu} (\boldsymbol{\nu} a)P'}\; a \notin \mathrm{n}(\mu) \qquad \frac{P \xrightarrow{\overline{a}b} P'}{(\boldsymbol{\nu} b)P \xrightarrow{\overline{a}(b)} P'}\; a \neq b$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\; \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset \qquad \frac{P \xrightarrow{\overline{a}b} P' \qquad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{\overline{a}(b)} P' \qquad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} b)(P' \mid Q')}\; b \notin \mathrm{fn}(Q)$$

Figure 2.1: Labelled transitions in the π-calculus ($\stackrel{\cdot}{\longrightarrow}_\pi$)

Choice and replication can be added exactly as for CCS: by adding rules from (1.4) and (2.6), respectively. The bisimulation has to be modified to handle extrusion:

Definition 2.2.3 (Early bisimilarity in π)**.** A relation $\mathcal{R}$ on π-calculus processes is a strong early bisimulation if, whenever $P\ \mathcal{R}\ Q$:

1. if $P \xrightarrow{\overline{a}(b)}_\pi P'$ and $b \notin \mathrm{fn}(Q)$ then $Q \xrightarrow{\overline{a}(b)}_\pi Q'$ for some Q' such that $P'\ \mathcal{R}\ Q'$,

2. if $P \xrightarrow{\mu}_\pi P'$ and μ is not a bound output, then $Q \xrightarrow{\mu}_\pi Q'$ for some Q' such that $P' \mathcal{R} Q'$,

3. the converse of (1) and (2), on Q.

Early bisimilarity, $\sim^{\mathrm{e}}$, is the union of all early bisimulations.

Using the same notation for weak transitions as in Definition 2.1.8, we get a notion of weak bisimilarity:

Definition 2.2.4 (Weak early bisimilarity in π)**.** A relation $\mathcal{R}$ on π-calculus processes is a weak early bisimulation if, whenever $P \mathcal{R} Q$:

1. if $P \xrightarrow{\overline{a}(b)}_\pi P'$ and $b \notin \mathrm{fn}(Q)$ then $Q \overset{\overline{a}(b)}{\Longrightarrow}_\pi Q'$ for some Q' such that $P' \mathcal{R} Q'$,

2. if $P \xrightarrow{\mu}_\pi P'$ and μ is not a bound output, then $Q \overset{\hat{\mu}}{\Longrightarrow}_\pi Q'$ for some Q' such that $P' \mathcal{R} Q'$,

3. the converse of (1) and (2), on Q.

Weak early bisimilarity, $\approx^{\mathrm{e}}$, is the union of all weak early bisimulations.

Note how inputs and outputs are treated differently, both in the reduction semantics, the transitions, and the bisimulation. If in CCS, one can trivially exchange inputs and outputs whilst essentially preserving the semantics, this does not seem simple in the π-calculus.

2.3 Typing and subtyping for the π-calculus

As in the sequential and functional world, types for the π-calculus are used to impose constraints on how programs or resources are used. They allow more precise behavioural analyses, and are often required to design scalable systems. Types for the π-calculus were first used to control arities of channels [Mil92] (in which case types are called *sorts*), and then were extended by Pierce and Sangiorgi [PS93] to capability types. The latter type discipline focuses on how channel names are used, ensuring that only some names will be used as receiving channels, and only some others as emitting channels. They also determine which kind of data can be exchanged through channels.

Types, ranged over by T, U,... are described by the following grammar:

$$T \ ::= \ \mathtt{i}T \ \mid \ \mathtt{o}T \ \mid \ \sharp T \ \mid \ \mathbf{1} \ \mid \ X \ \mid \ \mu X.T \tag{2.10}$$

where $\mathtt{i}T$, called an *input type*, is the capability of receiving data of type T. *Output type* $\mathtt{o}T$ is the capability of sending such data. Names with type $\sharp T$ have both capabilities. The *unit type* is written $\mathbf{1}$ and provides no capability. We allow recursive definitions and consider equal $\mu X.T$ and $T\{\mu X.T/X\}$ to be able to manipulate infinite, regular types.

We refer to those types as input/output types, or i/o-types. Note that data carried over in the plain π-calculus consist only of channel names, but i/o-types are well-suited to extensions to other kinds of data [SW01]. Rules for i/o-types are listed in Figure 2.2. Note that in case of ambiguity (mainly in Section 3.3), the resulting typed calculus is written π^{io}.

We use Γ, Δ to range over type environments, i.e., partial functions from names to types. We write $\mathsf{dom}(\Gamma)$ for the domain of Γ, that is, the set of names on which Γ is defined. In name-passing calculi, a type system assigns a type to each name. Typing judgements are of the form $\Gamma \vdash P$ (process P respects the type assignments in Γ), and $\Gamma \vdash a : T$ (name a can be assigned type T in Γ).

Note that rule T-NAME mentions a subtyping relation $\leq$. Subtyping is very natural for capability types: some names can have more capabilities than others. Relation $\leq$ is defined inductively[2] in Figure 2.3: we see here that capability $\mathtt{i}$ is covariant, that $\mathtt{o}$ is contravariant, and that $\sharp$ has no special variance. Rules SUB-$\sharp\mathtt{i}$ and SUB-$\sharp\mathtt{o}$ formally say that capability $\sharp$ subsumes the other two.

[2]Whilst this definition is enough for our needs, it does not cover all the cases for recursive types. For a satisfactory characterisation we can say that $\leq$ is in fact defined *coinductively* through the same set of rules and modulo $\mu X.T = T\{\mu X.T/X\}$. For a less cumbersome way, we refer to the different set of rules explicitly taking into account recursive definitions, in the original work [PS93].

$$\text{T-NAME}\ \frac{\Gamma(a) \leq T}{\Gamma \vdash a : T} \qquad \text{T-NIL}\ \frac{}{\Gamma \vdash 0} \qquad \text{T-PAR}\ \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \text{T-REP}\ \frac{\Gamma \vdash P}{\Gamma \vdash\ !P} \qquad \text{T-RES}\ \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} a)P}$$

$$\text{T-OUT}\ \frac{\Gamma \vdash a : \mathtt{o}T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}\langle b \rangle.P} \qquad \text{T-INP}\ \frac{\Gamma \vdash a : \mathtt{i}T \qquad \Gamma, x : T \vdash P}{\Gamma \vdash a(x).P}$$

Figure 2.2: i/o-types in the π-calculus (written π^{io})

$$\text{SUB-REFL}\ \frac{}{T \leq T} \qquad \text{SUB-TRANS}\ \frac{T_1 \leq T_2 \qquad T_2 \leq T_3}{T_1 \leq T_3} \qquad \text{SUB-}\sharp\mathtt{i}\ \frac{}{\sharp T \leq \mathtt{i}T} \qquad \text{SUB-}\sharp\mathtt{o}\ \frac{}{\sharp T \leq \mathtt{o}T} \qquad \text{SUB-ii}\ \frac{T_1 \leq T_2}{\mathtt{i}T_1 \leq \mathtt{i}T_2} \qquad \text{SUB-oo}\ \frac{T_1 \leq T_2}{\mathtt{o}T_2 \leq \mathtt{o}T_1}$$

Figure 2.3: Subtyping for i/o-types in π

Subtyping can be directly related to typing judgements: when $T_1 \leq T_2$, we know that T_1 can be used in place of T_2 in every typing judgement. We call this the narrowing property:

Lemma 2.3.1 (Narrowing for π)**.** *Suppose $T_1 \leq T_2$. Then $\Gamma, a : T_2 \vdash P$ implies $\Gamma, a : T_1 \vdash P$.*

Narrowing—especially written as it is above—says that the subsumption rule:

$$\text{SUBSUMPTION}\ \frac{T_1 \leq T_2 \qquad \Gamma, a : T_2 \vdash P}{\Gamma, a : T_1 \vdash P}$$

is admissible. It is sometimes included in the type system, with a more strict version of T-NAME; the two presentations are strictly equivalent. It can be used to derive the preservation of the typing information through reduction:

Lemma 2.3.2 (Subject reduction for π)**.** *For every π-calculus process P, if $\Gamma \vdash P$ and $P \longrightarrow_\pi P'$ then $\Gamma \vdash P'$.*

Subject reduction, together with the shape of the typing rules, let one derive in turn Corollary 2.3.3, a safety property about how names can be used: e.g. if a name is typed with an input type, then it will not be used as an output.

Corollary 2.3.3 (Type consistency for π)**.** *If $\Gamma, a : \mathtt{i}T \vdash P$ then $P \not\Downarrow^\pi_{\overline{a}}$ and if if $\Gamma, a : \mathtt{o}T \vdash P$ then $P \not\Downarrow^\pi_a$.*

Polyadic π-calculus [Mil92] We mentioned in Section 1.3 that the polyadic π-calculus could be encoded in the regular (monadic) π-calculus. While this indeed means that polyadic and monadic π have the same raw expressive power, this does not mean that the types can be preserved through this encoding, and in fact they are more informative in the polyadic variant. In this paragraph, we elaborate on this. The grammar of the polyadic π-calculus is as follows:

$$P \ ::= \ 0 \ \mid \ P \mid Q \ \mid \ !P \ \mid \ (\boldsymbol{\nu} a)P \ \mid \ \overline{a}\langle b_1, \ldots, b_n \rangle.P \ \mid \ a(x_1, \ldots, x_m).P \ \mid \ \mathtt{wrong}\ ,$$

the base case of the reduction relation in π is replaced with the following [PS93]:

$$\overline{a}\langle b_1, \ldots, b_n \rangle.P \mid a(x_1, \ldots, x_m).Q \longrightarrow \begin{cases} P \mid Q\{b_1/x_1\} \ldots \{b_n/x_m\} & \text{if } m = n \\ \mathtt{wrong} & \text{if } m \neq n \end{cases},$$

and structural congruence is extended with $\texttt{wrong} \equiv (\boldsymbol{\nu} a)\texttt{wrong} \equiv P \mid \texttt{wrong} \equiv !\texttt{wrong}$. The construct `wrong` emphasises the fact that there might be undesirable arity mismatches. Milner introduced [Mil93] a sorting system to prevent those, but here we directly present the extension to i/o-types [PS93]. Now, types can be grouped into tuples:

$$T \;::=\; c(T_1, \ldots, T_n) \;\mid\; \mathbf{1} \;\mid\; X \;\mid\; \mu X.T \qquad\qquad c \;::=\; \mathtt{i} \;\mid\; \mathtt{o} \;\mid\; \sharp$$

and typing rules are straightforward but more verbose:

$$\frac{\Gamma \vdash a : \mathtt{o}(T_1, \ldots, T_n) \qquad (\Gamma \vdash b_i : T)_{\forall i \in \{1,\ldots,n\}} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}\langle b_1, \ldots, b_n\rangle.P} \qquad\qquad \frac{\Gamma \vdash a : \mathtt{i}(T_1, \ldots, T_n) \qquad \Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash P}{\Gamma \vdash a(x_1, \ldots, x_n).P} \;.$$

Subject reduction holds for this type system. Consequently, since there is no rule to derive $\Gamma \vdash \texttt{wrong}$, we know that if $\Gamma \vdash P$ and $P \Longrightarrow_\pi P'$ then $P' \not\equiv \texttt{wrong}$.

In principle, polyadic types are not more intricate than the ones for the monadic π-calculus, but the corresponding type discipline is expressive and useful in practice (note that the usual encoding from polyadic π to monadic π does not preserve types). In light of this, in this work, we handle polyadic prefixes only when needed or as extensions, and not in the base grammar.

Notation When dealing with monadic calculi, we will sometimes omit the chevrons $\langle\,\rangle$ in prefixes, writing e.g. $\overline{a}b$ instead of $\overline{a}\langle b\rangle$.

2.4 The λ-calculus in the π-calculus

Call-by-name The λ-calculus is the foundational calculus for functional programming, and is often used as a test of expressiveness for process calculi. We give below a call-by-name flavour of λ-calculus [Plo75]—more precisely the weak head reduction strategy—which proved to be simple to encode. The set Λ of pure λ-terms is defined by:

$$M, N \;::=\; x \;\mid\; \lambda x.M \;\mid\; MN$$

where x is bound in $\lambda x.M$. We write Λ^0 for the subset of closed terms. The *call-by-name reduction (cbn) relation* $\longmapsto_{\mathsf{n}}$ is the least relation over Λ^0 that is closed under the following rules:

$$\frac{}{(\lambda x.M)N \longmapsto_{\mathsf{n}} M\{N/x\}} \qquad\qquad \frac{M \longmapsto_{\mathsf{n}} M'}{MN \longmapsto_{\mathsf{n}} M'N}$$

We write $\Longmapsto_{\mathsf{n}}$ for the reflexive and transitive closure of $\longmapsto_{\mathsf{n}}$. The values are the terms of the form $\lambda x.M$ and are ranged by V, W. In call-by-name, *evaluation contexts* are described by the following grammar:

$$C \;::=\; CM \;\mid\; [\cdot]$$

Symbol C is used also for arbitrary contexts; it will be explicitly indicated when C refers to an evaluation context.

Call-by-value Plotkin [Plo75] also focuses on a variant of λ where the beta-reduction only happens when the argument is a value, giving birth to a significantly different equational theory. We use what is called the left-to-right call-by-value (cbv), written $\longmapsto_{\mathsf{v}}$, and given by the following rules:

$$\frac{}{(\lambda x.M)V \longmapsto_{\mathsf{v}} M\{V/x\}} \qquad \frac{M \longmapsto_{\mathsf{v}} M'}{MN \longmapsto_{\mathsf{v}} M'N} \qquad \frac{N \longmapsto_{\mathsf{v}} N'}{VN \longmapsto_{\mathsf{v}} VN'} \;.$$

We write $\Longmapsto_{\mathsf{v}}$ for the reflexive and transitive closure of $\longmapsto_{\mathsf{v}}$. We can deduce that *evaluation contexts* in call-by-value can be described by the following grammar:

$$C \;::=\; CN \;\mid\; VC \;\mid\; [\cdot]$$

Encoding cbv in pi As an example of an encoding result, we recall an encoding of call-by-value λ into π given in [SW01]. Note that values are represented by names ranged over by v, and that the encoding $[\![M]\!]_p$ of a term M can be handled to the environment through the name p, which we call the handle.

$$\begin{aligned}[\![\lambda x.M]\!]_p &= (\boldsymbol{\nu} v)\overline{p}v.!v(x,q).[\![M]\!]_q \\ [\![x]\!]_p &= \overline{p}x \\ [\![MN]\!]_p &= (\nu q)\big([\![M]\!]_q \mid q(v).(\nu r)([\![N]\!]_r \mid r(w).\overline{v}\langle w,p\rangle)\big) \ .\end{aligned}$$

To understand this encoding, looking at a cbv beta-redex (a term of the form V_1V_2) is insightful. For example the encoding of the term $(\lambda x.M)(\lambda y.N)$ reduces in a deterministic way to a process that behaves like $[\![M]\!]_p$ except that x is bound to a server that provides N whenever x is called:

$$[\![(\lambda x.M)(\lambda y.N)]\!]_p \approx (\nu x)\big([\![M]\!]_p \mid !x(y,q).[\![N]\!]_q\big) \ .$$

We can see how process $!x(y,q).[\![N]\!]_q$ behaves as an environment mapping x to $\lambda y.N$, while the restriction on x is preventing an outside process from tampering with the mapping. (One could also view it as an explicit substitution $(\lambda y.N/x)$, as explicit substitutions [ACCL90] are decomposing the usual beta-reduction into more elementary steps.) Note that thanks to the replication in the server, x can be used multiple times in M.

There are simpler encodings, especially for call-by-name, where the analogy between environments and servers is more immediate. However, more can be said about the cbv encoding above, with respect to the typing discipline:

- a first version of this encoding was studied in Milner's Functions as Processes [Mil90] with a more verbose clause for the variable: $[\![x]\!]_p = (\boldsymbol{\nu} v)\overline{p}v.!v(y,q).\overline{x}\langle y,q\rangle$. As Milner remarked, computations can take more time with this encoding (for instance, each beta step of the usual diverging term $\Omega = (\lambda x.xx)(\lambda x.xx)$ takes an increasing number of silent reductions in π, instead of just one with the optimised encoding).

  It took the typing system introduced by Pierce and Sangiorgi [PS93] to prove the soundness of the optimised encoding.

- The types of the simply-typed lambda-calculus are preserved by this encoding; we recall here the typing rules, building arrow types on top of base types (the latter being ranged over by ι):

  $$A \ ::= \ \iota \mid A \to B \qquad \frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

  a λ-type A corresponds to a π-type $[A]$ defined as:

  $$[\iota] \triangleq \iota \qquad\qquad [A \to B] \triangleq \mathsf{o}([A], \mathsf{o}[B]) \ . \tag{2.11}$$

  The correspondence for typing judgements ([SW01], Corollary 16.2.3) goes as follows: types in the environment are encoded as above, and whilst a term M can have a type B, process $[\![M]\!]_p$ cannot: it can only be well-typed. The typing information B is transferred to the handle p:

  $$\begin{array}{lll} \text{if} & x_1 : A_1 \ , \ \ldots \ , x_n : A_n & \vdash \ M \ : B \\ \text{then} & x_1 : [A_1], \ \ldots \ , x_n : [A_n], \ p : \mathsf{oo}[B] & \vdash [\![M]\!]_p \ . \end{array}$$

- ([SW01], Remarks 16.2.5 and 16.2.6) The relation between types in lambda- and pi-calculi is robust. Extending lambda-calculus with subtyping is done by remarking that $A \to B$ is contravariant in A and covariant in B, which is the same for $[A \to B]$ in (2.11). If we extend lambda with a type D such that D is equal to $D \to D$, then every term can be typed with type D. Interestingly the solution to $[D] = [D \to D]$ (i.e. $[D] = \mu X.\mathsf{o}(X, \mathsf{o}X)$ allowed by recursive types in π) corresponds to the original type of the encoding [PS93].

Chapter 3

Symmetric calculi

In this chapter, we first recall some symmetric calculi known in the literature. We already remarked that CCS is symmetric: it is easy to see that swapping inputs and outputs in processes does not change the induced equivalence. To say this formally, one can define a dualisation operator, mapping a process P to its dual $\overline{P}$, defined as follows:

Definition 3.0.1 (Dual of a process in CCS)**.** The dual of a process P, written $\overline{P}$ is defined by the following recursive definition:

$$\overline{\overline{a}.P} \triangleq a.\overline{P} \qquad \overline{a.P} \triangleq \overline{a}.\overline{P} \qquad \overline{0} \triangleq 0 \qquad \overline{P \mid Q} \triangleq \overline{P} \mid \overline{Q} \qquad \overline{(\boldsymbol{\nu} a)P} \triangleq (\boldsymbol{\nu} a)\overline{P} \ .$$

The first two equations state that duality indeed swaps inputs and outputs; we say that the last three make the dual operator homomorphic with respect to the other operators of the calculus. We observe the following straightforward symmetry properties in CCS:

Proposition 3.0.2. *For all CCS processes P and Q, $P \longrightarrow Q$ if and only if $\overline{P} \longrightarrow \overline{Q}$.*

Proposition 3.0.3. *For all CCS processes P and Q, $P \simeq Q$ if and only if $\overline{P} \simeq \overline{Q}$.*

In the following, we describe more elaborate calculi having similar properties.

3.1 Symmetry as internal mobility

There is a significant expressiveness gap between CCS and the π-calculus: the former has a cleaner theory, the latter allowing more, and cleaner, encodings of data types. The π-calculus with internal mobility [San96b], written $\pi\mathrm{I}$, forms a bridge between the two: the behavioural theory is simple, and the calculus is expressive. One way to define $\pi\mathrm{I}$ is to consider the subcalculus of π where all emitted data are always private names, i.e. outputs are always of the form $(\boldsymbol{\nu} x)\overline{a}\langle x\rangle.P$, which is generally written $\overline{a}(x).P$, as x is bound. We know that $\pi\mathrm{I}$ is indeed a subcalculus of π, i.e.:

$$\pi\mathrm{I} \subseteq \pi \qquad\qquad (\pi\mathrm{I}\longrightarrow) \subseteq (\equiv \pi\mathrm{I}) \ .$$

The first inclusion tells us that $\pi\mathrm{I}$ is a subset of π; the second inclusion tells us that $\pi\mathrm{I}$ is a calculus by itself, i.e. that $\pi\mathrm{I}$ is closed by reduction (up to structural congruence: if $P \in \pi\mathrm{I}$ and $P \longrightarrow Q$ then $Q \equiv Q'$ for some $Q' \in \pi\mathrm{I}$). Whilst this calculus fits the presentation we gave in Section 1.1, we prefer to use a dedicated grammar:

Definition 3.1.1. Terms of $\pi\mathrm{I}$ are generated by the following grammar:

$$P \ ::= \ 0 \ \mid \ P \mid Q \ \mid \ (\boldsymbol{\nu} a)P \ \mid \ \overline{a}(x).P \ \mid \ a(x).P$$

where x is bound in $(\boldsymbol{\nu}x)P$, $\overline{a}(x).P$ and $a(x).P$. Reduction in πI, written $\longrightarrow_{\pi\mathrm{I}}$, is the least relation $\longrightarrow$ generated by the following rules:

$$\frac{}{\overline{a}(x).P \mid a(x).Q \longrightarrow (\boldsymbol{\nu}x)(P \mid Q)} \qquad \frac{P \longrightarrow P'}{P \mid R \longrightarrow P' \mid R} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu}a)P \longrightarrow (\boldsymbol{\nu}a)P'} \qquad \frac{P \equiv \longrightarrow \equiv P'}{P \longrightarrow P'}$$

(Structural congruence is defined as usual, with the same laws as in Definition 2.1.1.) We give the corresponding labels—note that both labels are binding ($\mathrm{bn}(\overline{a}(x)) = \mathrm{bn}(a(x)) = \{x\}$):

$$\mu \quad ::= \quad a(x) \;\mid\; \overline{a}(x) \;\mid\; \tau$$

and LTS in Figure 3.1. Remark the symmetry of the communication rule.

$$\frac{}{\overline{a}(x).P \xrightarrow{\overline{a}(x)} P} \qquad \frac{}{a(x).P \xrightarrow{a(x)} P}$$

$$\frac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu}aP \xrightarrow{\mu} \boldsymbol{\nu}aP'}\ a \notin \mathrm{n}(\mu) \qquad \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset \qquad \frac{P \xrightarrow{\overline{a}(x)} P' \qquad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}x)(P' \mid Q')}$$

Figure 3.1: Labelled transitions in πI ($\dot{\longrightarrow}_{\pi\mathrm{I}}$)

Note that guarded sum and recursive definitions are usually included in πI. We do not present them here, for the sake of uniformity, but recursive definitions are useful to provide πI with its full expressiveness: it can encode, for example, the λ-calculus [San96b]. As in CCS, an input/output duality operator can be defined on πI terms:

$$\overline{\overline{a}(x).P} \triangleq a(x).\overline{P} \qquad \overline{a(x).P} \triangleq \overline{a}(x).\overline{P} \qquad \overline{0} \triangleq 0 \qquad \overline{P \mid Q} \triangleq \overline{P} \mid \overline{Q} \qquad \overline{(\boldsymbol{\nu}a)P} \triangleq (\boldsymbol{\nu}a)\overline{P}$$

and the properties of symmetry of CCS (Propositions 3.0.2 and 3.0.3) can also be derived in πI.

3.2 Symmetry in fusion calculi

The π-calculus has two kinds of prefixes: output prefixes, with free objects and input prefixes, with bound objects. We say that "outputs are free" and that "inputs are bound". The πI-calculus gets rid of this asymmetry by having both bound outputs and bound inputs. This suggests that there could be an approach in the opposite direction, with both free outputs and free inputs. Moreover, this approach removes the binding role from the prefix operator, which serves minimality: the resulting calculus would feature only one binder. In fact, this yields a variety of calculi, that we call here fusion calculi. We give in this section a description of three of them: the update calculus in Section 3.2.1, the fusion calculus in Section 3.2.2 and the explicit fusion calculus in Section 3.2.3.

The first question asked when designing such calculi is what happens during interactions: the objects must be made to be related somehow. In π, the bound object is replaced by the free object. In πI, both bound objects are unified, through α-conversion. In fusion calculi, how objects are related will determine the calculus itself:

	Output	Input	Reduction		
πI	$\overline{a}(x).P$	$a(x).Q$	$\overline{a}(x).P \mid a(x).Q$	$\longrightarrow$	$(\boldsymbol{\nu}x)(P \mid Q)$
π	$\overline{a}\langle b\rangle.P$	$a(x).Q$	$\overline{a}\langle b\rangle.P \mid a(x).Q$	$\longrightarrow$	$P \mid Q\{b/x\}$
fusion calculi	$\overline{a}\langle b\rangle.P$	$a\langle c\rangle.Q$	$\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q$	$\longrightarrow$	$?$

We describe below the main flavours of fusion calculi, that answer this question in three different ways.

3.2.1 The update calculus

By analogy with the π-calculus, replacing the input object with the output object seems to be the natural option. In a fusion calculus, to do so in a sensible manner, one must replace all occurrences of the name in input object. Whilst this operation was easy to define in the π-calculus, the fact that the binder is not tied to the input prefix anymore makes it not well defined in all cases. The first solution, used in the update-calculus [PV97], is to apply the substitution within the whole scope of a name, where the scope is delimited by the restriction operator. Hence, a substitution of a name c with a different name b only takes place inside a process of the form $(\boldsymbol{\nu} c)R$ and will result in $R\{b/c\}$. Formally, grammar and reduction are defined as follows:

$$P \;::=\; 0 \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \overline{a}\langle b\rangle.P \;\mid\; a\langle b\rangle.P$$

$$\begin{aligned} \overline{a}\langle b\rangle.P \mid a\langle b\rangle.Q &\longrightarrow_{\mathrm{up}} P \mid Q \\ (\boldsymbol{\nu} c)(\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q \mid R) &\longrightarrow_{\mathrm{up}} (P \mid Q \mid R)\{b/c\} \end{aligned}$$

In addition, as usual, $\longrightarrow$ is closed by parallel contexts, restriction, and structural congruence, the latter being defined with again the rules from Definition 2.1.1.

If the syntax is symmetric, the reduction relation is not, contrarily to CCS and πI. Indeed, $(\boldsymbol{\nu} c)(\overline{a}\langle b\rangle.0 \mid a\langle c\rangle.0) \longrightarrow_{\mathrm{up}} 0$ but the symmetric process $(\boldsymbol{\nu} c)(a\langle b\rangle.0 \mid \overline{a}\langle c\rangle.0)$ has no reduction through $\longrightarrow_{\mathrm{up}}$. Whilst this itself is not a problem, the update calculus has behaviours one might not expect at first glance:

- **two-way substitutions:** it seems that substitution is going one way only: input objects are being replaced with output objects. However, a combination of two input objects might induce a replacement of an output object:

  $$\begin{aligned} (\boldsymbol{\nu} cb)(\overline{u}\langle b\rangle \mid u\langle c\rangle \mid \overline{v}\langle a\rangle \mid v\langle c\rangle \mid P) &\longrightarrow_{\mathrm{up}} (\boldsymbol{\nu} b)(\overline{v}\langle a\rangle \mid v\langle b\rangle \mid P\{b/c\}) \\ &\longrightarrow_{\mathrm{up}} P\{b/c\}\{a/b\} \;\; . \end{aligned}$$

  In the reduction above, name b is replaced with name a, whereas b appears as output object in the first process.

  This means that we cannot statically ensure that output objects are protected from replacement, which impedes the detection of future aliasing of two names.

  In addition to not knowing if two names might become equal in the future, this forbids some forms of type systems in the update-calculus (Section 3.4).

- **polyadicity:** the update calculus lacks robustness: as remarked in [PV97], considering its polyadic version raises the question "which process, if any, should $(\boldsymbol{\nu} d)(\overline{a}\langle b, c\rangle \mid a\langle d, d\rangle)$ reduce to?". This suggests again that output objects may be altered along the reduction.

We present next the fusion calculus, which solves the second observation by making reduction symmetric, and takes two-way substitutions as a feature instead of an observation. In Chapter 4, we present a calculus that can be seen as a corrected variant of the update-calculus, with, indeed, one-way substitutions.

3.2.2 The fusion calculus

The fusion calculus [PV98a], occasionally called *implicit* fusion calculus (or just "implicit fusions") to avoid confusion with Section 3.2.3, is the symmetrised version of the update calculus where objects of both polarities can be subject to substitution in the scope of a restriction. Its grammar is the same as in the update calculus:

$$P \;::=\; 0 \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \overline{a}\langle b\rangle.P \;\mid\; a\langle b\rangle.P$$

and its reductions are defined below (closed under $\equiv$ and the usual contexts):

$$\begin{aligned}
\overline{a}\langle b\rangle.P \mid a\langle b\rangle.Q \quad &\longrightarrow_{\mathrm{fu}} \quad P \mid Q \\
(\boldsymbol{\nu} c)(\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q \mid R) &\longrightarrow_{\mathrm{fu}} (P \mid Q \mid R)\{b/c\} \\
(\boldsymbol{\nu} b)(\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q \mid R) &\longrightarrow_{\mathrm{fu}} (P \mid Q \mid R)\{c/b\}
\end{aligned}$$

The usual duality operator defined as $\overline{\overline{a}\langle b\rangle.P} \triangleq a\langle b\rangle.\overline{P}$ and $\overline{a\langle b\rangle.P} \triangleq \overline{a}\langle b\rangle.\overline{P}$ generates the same symmetry correspondence as in CCS, in Proposition 3.0.3. The polyadic variant can be described using a single rule:

$$(\boldsymbol{\nu} d_1, \ldots, d_m)(\overline{a}\langle b_1, \ldots, b_n\rangle.P \mid a\langle c_1, \ldots, c_n\rangle.Q \mid R) \longrightarrow_{\mathrm{fu}} (P \mid Q \mid R)\sigma \ ,$$

is a reduction when σ is a name substitution, such that:

- the range of σ is $\{b_1, \ldots, b_n, c_1, \ldots, c_n\} \setminus \{d_1, \ldots, d_m\}$
- the domain of σ is $\{d_1, \ldots, d_m\}$,
- and such that $\sigma(b_i) = \sigma(c_i)$ for all $i \leq n$.

(The domain of σ is the set $\{x \mid x \neq \sigma(x)\}$, and the range of σ is $\{\sigma(x) \mid x \neq \sigma(x)\}$.)

The reductions presented above are not compositional, mainly because the substitution happens in the whole scope of the restriction. Another presentation [PV98a], which is used to build a compositional LTS for the implicit fusion calculus, puts the substitution on hold, waiting for a potential restriction from the environment:

$$\frac{P \xrightarrow{\overline{a}b}_{\mathrm{fu}} P' \qquad Q \xrightarrow{ac}_{\mathrm{fu}} Q'}{P \mid Q \xrightarrow{\{b=c\}}_{\mathrm{fu}} P' \mid Q'} \qquad \frac{P \xrightarrow{\{b=c\}}_{\mathrm{fu}} P'}{(\boldsymbol{\nu} c)P \xrightarrow{\mathbf{1}}_{\mathrm{fu}} P'\{b/c\}} \qquad \frac{P \xrightarrow{\{b=c\}}_{\mathrm{fu}} P'}{(\boldsymbol{\nu} b)P \xrightarrow{\mathbf{1}}_{\mathrm{fu}} P'\{c/b\}}$$

The labels can be, in addition to output and input labels, equivalence relations on names. A transition labelled with $\mathbf{1}$ (the identity relation) is a silent transition, and $\{b{=}c\}$ represents the smallest equivalence relation containing (b, c). For example, $\{b{=}b\} = \mathbf{1}$.

Other rules, described in details in [PV98a], complete the set, but the ones above serve our introductory purposes.

3.2.3 The explicit fusion calculus

The explicit fusion calculus [GW00] transforms the transition $P \xrightarrow{\{b=c\}}_{\mathrm{fu}} P'$ from the implicit fusion calculus into a real silent transition and postpones the act equating of b and c. It relates b in c in an explicit manner, in a process construct $b{=}c$, called a *fusion*, resulting in the reduction $P \longrightarrow_{\mathrm{ef}} P' \mid b{=}c$.

$$P \xrightarrow{\{b=c\}}_{\mathrm{fu}} P' \quad \rightsquigarrow \quad P \longrightarrow_{\mathrm{ef}} P' \mid b{=}c \ .$$

The role of the fusion $b{=}c$ is to somehow equate b and c in the upcoming reductions of the process. We give more details about this calculus. First, the grammar of the explicit fusion calculus (or "explicit fusions") is the same as the one for the implicit fusion calculus, with the additional fusion construct:

$$P \ ::= \ 0 \ \mid \ P \mid Q \ \mid \ (\boldsymbol{\nu} a)P \ \mid \ \overline{a}\langle b\rangle.P \ \mid \ a\langle b\rangle.P \mid a{=}b \ .$$

The base case of the reduction $\longrightarrow_{\mathrm{ef}}$ appeals by its simplicity:

$$\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q \longrightarrow_{\mathrm{ef}} P \mid Q \mid b{=}c \ ,$$

and can be extended naturally to polyadic communication:

$$\overline{a}\langle b_1, \ldots, b_n\rangle.P \mid a\langle c_1, \ldots, c_n\rangle.Q \longrightarrow_{\mathrm{ef}} P \mid Q \mid b_1{=}c_1 \mid \ldots \mid b_n{=}c_n \ .$$

However, to give meaning to the fusion construct, structural congruence from Definition 2.1.1 has to be augmented with rewriting rules:

$$P \mid a{=}b \equiv P\{b/a\} \mid a{=}b \qquad a{=}a \equiv 0 \qquad (\boldsymbol{\nu} a)a{=}b \equiv 0 \ . \tag{3.1}$$

Note that given the laws above, the laws about symmetry ($a{=}b \equiv b{=}a$), transitivity ($a{=}b \mid b{=}c \equiv a{=}c \mid b{=}c$), and 'selective rewriting' ($a{=}b \mid P\{a/x\} \equiv a{=}b \mid P\{b/x\}$) can be derived.

Compositional LTS Alternatively, one can give a compositional presentation of the explicit fusion calculus without mentioning structural congruence. Whilst the implicit fusion calculus needs a restriction $\boldsymbol{\nu} c$ or $\boldsymbol{\nu} d$ to trigger the reduction between $\overline{a}\langle c\rangle.P$ and $a\langle d\rangle.Q$, the explicit fusion calculus needs a fusion $a{=}b$ to trigger the reduction between $\overline{a}\langle c\rangle.P$ and $b\langle d\rangle.Q$.

Because of this, the labels of the LTS must anticipate such fusions, which can come from the environment. They are ranged by μ and can be either outputs, inputs, or conditional τs:

$$\mu \ ::= \ \overline{a}(x) \mid a(x) \mid [a=b]\tau \ ,$$

where x is bound in $\overline{a}(x)$ and $a(x)$. The action $[a{=}b]\tau$ means that a reduction can happen if the environment is able to equate a and b. We therefore identify τ with labels of the form $[a=a]\tau$ where a can be any name.

We give below *a variant of* the original LTS from Wischik [WG04].

The judgement $P \rhd a = b$, read as "P enforces $a = b$", is defined by the following rules:

$$\frac{}{P \rhd a = a} \qquad \frac{P \rhd a = b}{P \rhd b = a} \qquad \frac{P \rhd a = b \qquad P \rhd b = c}{P \rhd a = c}$$

$$\frac{}{a{=}b \rhd a = b} \qquad \frac{P \rhd a = b}{P \mid Q \rhd a = b} \qquad \frac{P \rhd a = b}{Q \mid P \rhd a = b} \qquad \frac{P \rhd a = b \qquad c \notin \{a, b\}}{(\boldsymbol{\nu} c)P \rhd a = b} \ .$$

The LTS defining the transitions $\dot{\longrightarrow}_{\mathrm{ef}}$ is defined by the rules of Figure 3.2. Note how the judgement $P \rhd a = b$ is only used in the last three rules to rewrite the labels.

$$\frac{}{\overline{a}\langle b\rangle.P \xrightarrow{\overline{a}(x)} b{=}x \mid P} \qquad \frac{}{a\langle b\rangle.P \xrightarrow{a(x)} b{=}x \mid P} \qquad \frac{P \xrightarrow{\overline{a}(x)} P' \qquad Q \xrightarrow{b(x)} Q'}{P \mid Q \xrightarrow{[a=b]\tau} (\boldsymbol{\nu} x)(P' \mid Q')}$$

$$\frac{P \xrightarrow{\mu} P' \qquad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q'} \qquad \frac{P \xrightarrow{\mu} P' \qquad a \notin \mathrm{n}(\mu)}{(\boldsymbol{\nu} a)P \xrightarrow{\mu} (\boldsymbol{\nu} a)P'}$$

$$\frac{P \xrightarrow{a(x)} P' \qquad P \rhd a = b}{P \xrightarrow{b(x)} P'} \qquad \frac{P \xrightarrow{\overline{a}(x)} P' \qquad P \rhd a = b}{P \xrightarrow{\overline{b}(x)} P'} \qquad \frac{P \xrightarrow{[a=b]\tau} P' \qquad P \rhd b = c}{P \xrightarrow{[c=a]\tau} P'} \ .$$

Figure 3.2: Labelled transitions in the explicit fusion calculus ($\dot{\longrightarrow}_{\mathrm{ef}}$)

This LTS coincides with the one from [WG04]) up to structural congruence; the differences are small: objects are handled using bound names (instead of concretions), and fusions are introduced to link objects to bound names. We focus on which names can be equated, instead of actually rewriting anything.

Given these definitions of LTS $\dot{\longrightarrow}_{\mathrm{ef}}$ and judgement $\rhd$, we can define the notion of efficient bisimulation [WG04] that captures barbed congruence.

Definition 3.2.1. A symmetric relation $\mathcal{R}$ is an efficient bisimulation if, whenever $P \mathcal{R} Q$:

- for all a and b, if $P \rhd a = b$ then $Q \rhd a = b$;
- if $P \xrightarrow{a(x)}_{\mathrm{ef}} P'$ and $x \notin \mathrm{fn}(Q)$, then $Q \xrightarrow{a(x)}_{\mathrm{ef}} Q'$ and $P' \mathcal{R} Q'$ for some Q';
- if $P \xrightarrow{\overline{a}(x)}_{\mathrm{ef}} P'$ and $x \notin \mathrm{fn}(Q)$, then $Q \xrightarrow{\overline{a}(x)}_{\mathrm{ef}} Q'$ and $P' \mathcal{R} Q'$ for some Q';
- if $P \xrightarrow{[a=b]\tau}_{\mathrm{ef}} P'$ then $(Q \mid a{=}b) \xrightarrow{\tau}_{\mathrm{ef}} Q'$ and $(P' \mid a{=}b) \mathcal{R} Q'$ for some Q'.

We write $\sim_{\mathrm{ef}}$ for the largest efficient bisimulation.

Remark 3.2.2. Processes of the implicit fusion calculus can be seen as processes of the explicit calculus; the latter is actually a conservative extension of the former, as the identity encoding induces a full abstraction [Wis01]. Therefore, we denote the two of them using the word "fusion calculi" or sometimes even just "fusions".

We show an important difference (already highlighted in [PV97]) between π and fusion calculi: in π with choice, we have $x \mid \overline{y} = x.\overline{y} + \overline{y}.x$ if we have the guarantee that x is not equal to y. This guarantee can be ensured by adding two restrictions $(\boldsymbol{\nu} xy)$ before, for example, sending x and y to the environment. This yields the interleaving law:

$$(\boldsymbol{\nu} xy)\overline{a}x.\overline{a}y.(x \mid \overline{y}) = (\boldsymbol{\nu} xy)\overline{a}x.\overline{a}y.(x.\overline{y} + \overline{y}.x) \ . \tag{3.2}$$

The law holds in the π-calculus for barbed congruence but not for fusion calculi (for update, fusion, or explicit fusion calculi) since the two names x and y can be equated by a outside process $ac.ac$, authorising a synchronisation in the left-hand side of the resulting processes:

$$c \mid \overline{c} \quad \text{versus} \quad c.\overline{c} + \overline{c}.c \ .$$

Of course, this is only valid in fusion calculi extended with sums (using a rule similar to (2.8)), but one can build a counterexample replacing sum with replication using Counterexample 5.8.2 from [SW01], which is another inequality exploiting the aliasing of two names to distinguish processes.

This example shows how private names can be equated in fusion calculi, whereas this is impossible in the π-calculus. We study this more thoroughly in Section 4.4. We contribute on explicit fusions in two other ways: in Section 4.6.3 we give an axiomatisation of the calculus, and more generally, Chapter 4 introduces and studies an asymmetric variant of the explicit fusion calculus.

3.3 Symmetry as a tool

Sections 3.2.1 to 3.2.3 are an account of calculi taken in the literature that have an intrinsic symmetry. In this section, we move to a new contribution (published in [HMS12]). We build a symmetric calculus on top of the π-calculus that is not a interesting by itself, but which can be used as a tool, to study behavioural properties of π-calculus processes.

Reasoning about processes usually involves proving behavioural equivalences, sometimes working in a typed setting with *typed behavioural equivalences*, that allow one to express constraints about the observations available to the context when comparing two processes. These constraints are generally expressed using the i/o type discipline (Section 2.3).

If dualities may reveal underlying structure and lead to simpler theories, they can also be used to relate different mathematical entities. We study here how duality can help relate processes through typed behavioural equivalences. As it is detailed in Section 3.4 existing symmetric versions of π (πI and fusion calculi) are not suitable for such a purpose: duality breaks with the addition of ordinary i/o-types.

In this section, in order to work in a setting that provides a form of duality and where i/o-types can be used, we introduce a calculus named $\overline{\pi}$ (Section 3.3.1). Intuitively, $\overline{\pi}$ is the minimal symmetrical conservative

extension of π with input/output types: it has constructs for free input and bound output (note that bound output is not seen as a derived construct in $\overline{\pi}$). In $\overline{\pi}$, we rely on substitutions as the main mechanism at work along interactions. To achieve this, we forbid interactions involving a free input and a free output: the type system rules out processes that use both kinds of prefixes on the same channel. Calculus $\overline{\pi}$ differs from fusion calculi in how it handles interactions between free prefixes, which we can summarise in the following table:

	Reduction
fusion calculus	$(\boldsymbol{\nu} b)(\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q) \quad \longrightarrow_{\mathrm{fu}} \quad (P \mid Q)\{c/b\}$
explicit fusion calculus	$\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q \quad \longrightarrow_{\mathrm{ef}} \quad P \mid Q \mid b{=}c$
$\overline{\pi}$	$\overline{a}\langle b\rangle.P \mid a\langle c\rangle.Q$ is not well-typed

Calculus $\overline{\pi}$ contains π, and any π process that can be typed using i/o-types can be typed in exactly the same way in $\overline{\pi}$. Moreover $\overline{\pi}$ contains a 'dualised' version of π: one can choose to use some channels in free input and bound output. For such channels, the typing rules intuitively enforce a 'contract' on the usage of the transmitted name *on the side of the emitter* (dually to the typing rules presented above). Usual duality properties, Propositions 3.0.2 and 3.0.3, hold in $\overline{\pi}$. We also show how $\overline{\pi}$ can be related to π, by translating $\overline{\pi}$ into a variant of the π-calculus with i/o-types in a fully abstract way (in Section 3.3.2). This result shows that π and $\overline{\pi}$ are rather close in terms of expressiveness.

In Section 3.3.1 we define a notion of typed barbed congruence in $\overline{\pi}$, which allows us to validate at a behavioural level the properties we have mentioned above: two processes are equivalent if and only if their duals are. It seems that no other existing calculus with i/o-types enjoys this form of duality for behaviours.

As an application of $\overline{\pi}$, its dualities, and its behavioural theory, we use $\overline{\pi}$ to relate two encodings of call-by-name λ-calculus (Section 3.3.3). The first one is the ordinary encoding by Milner [Mil90], the second one is by van Bakel and Vigliotti [vBV09]. The two encodings are syntactically quite different. Milner's is *input-based*, in that an abstraction interacts with its environment via an input. In contrast, van Bakel and Vigliotti's is *output-based*. Moreover, only the latter makes use of *link processes*, that is, forwarders that under certain conditions act as substitutions.

3.3.1 A symmetric π-calculus with types

The syntax of $\overline{\pi}$ is defined with the following grammars:

$$P \;::=\; 0 \;\mid\; P \mid Q \;\mid\; !P \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \alpha.P \qquad\qquad \alpha ::= \rho\langle b\rangle \mid \rho(x) \qquad\qquad \rho ::= a \mid \overline{a}$$

$\overline{\pi}$ differs from the usual π-calculus by the presence of the free input $a\langle b\rangle$ and bound output $\overline{a}(x)$ prefixes. Note that in $\overline{\pi}$, the latter is *not* a notation for $(\boldsymbol{\nu} x)\overline{a}\langle x\rangle.P$, but a primitive construct. These prefixes are the symmetric counterpart of $\overline{a}\langle b\rangle$ and $a(x)$ respectively. Given a process P, $\mathrm{fn}(P)$ stands for the set of free names of P, given that restriction, bound input and bound output are binding constructs. Given ρ of the form a or $\overline{a}$, $\mathrm{n}(\rho)$ is defined by $\mathrm{n}(\overline{a}) = \mathrm{n}(a) = a$.

Structural congruence is standard, and defined as in CCS in Definition 2.1.1 (in particular, as it is usually the case, there are no axioms involving prefixes), with the addition of $!P \equiv !P \mid P$ (seen in (2.6)) for replication.

Definition 3.3.1 (Reduction in $\overline{\pi}$)**.** The reduction laws $\longrightarrow$ (sometimes written $\longrightarrow_{\overline{\pi}}$) allow communication involving two prefixes *only if at least one of them is bound*:

$$\frac{}{\overline{a}\langle b\rangle.P \mid a(x).Q \longrightarrow P \mid Q\{b/x\}} \qquad \frac{}{a\langle b\rangle.P \mid \overline{a}(x).Q \longrightarrow P \mid Q\{b/x\}} \qquad \frac{}{\overline{a}(x).P \mid a(x).Q \longrightarrow (\boldsymbol{\nu} x)(P \mid Q)}$$

$$\frac{P \longrightarrow P'}{P \mid R \longrightarrow P' \mid R} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} a)P \longrightarrow (\boldsymbol{\nu} a)P'} \qquad \frac{P \equiv \longrightarrow \equiv P'}{P \longrightarrow P'}$$

Note that $\overline{a}\langle b\rangle \mid a\langle c\rangle$ is a process of $\overline{\pi}$ that has no reduction; this process is ruled out by the type system presented in Figure 3.4.

Types and Behavioural Equivalence Types are a refinement of standard i/o-types: in addition to capabilities (ranged over using c), we annotate types with *sorts* (s), that specify whether a name can be used in free input (sort $\mathbf{e}$) or in free output ($\mathbf{r}$)—note that a name cannot be used to build both kinds of free prefixes.

$$T ::= c^s T \mid \mathbf{1} \qquad\qquad c ::= \mathtt{i} \mid \mathtt{o} \mid \sharp \qquad\qquad s ::= \mathbf{e} \mid \mathbf{r}$$

If name a has type $c^{\mathbf{r}}T$, we shall refer to a as an $\mathbf{r}$-*name*, and similarly for $\mathbf{e}$.

The subtyping relation is the smallest reflexive and transitive relation $\leq$ satisfying the rules of Figure 3.3. As in the π-calculus $\mathtt{i}^{\mathbf{r}}$ is covariant and $\mathtt{o}^{\mathbf{r}}$ is contravariant. Dually, $\mathtt{i}^{\mathbf{e}}$ is contravariant and $\mathtt{o}^{\mathbf{e}}$ is covariant. Note that sorts ($\mathbf{e}$, $\mathbf{r}$) are not affected by subtyping.

$$\frac{}{\sharp^s T \leq \mathtt{i}^s T} \qquad \frac{}{\sharp^s T \leq \mathtt{o}^s T} \qquad \frac{T_1 \leq T_2}{\mathtt{i}^{\mathbf{r}} T_1 \leq \mathtt{i}^{\mathbf{r}} T_2} \qquad \frac{T_1 \leq T_2}{\mathtt{o}^{\mathbf{r}} T_2 \leq \mathtt{o}^{\mathbf{r}} T_1} \qquad \frac{T_1 \leq T_2}{\mathtt{i}^{\mathbf{e}} T_2 \leq \mathtt{i}^{\mathbf{e}} T_1} \qquad \frac{T_1 \leq T_2}{\mathtt{o}^{\mathbf{e}} T_1 \leq \mathtt{o}^{\mathbf{e}} T_2}$$

Figure 3.3: Subtyping in $\overline{\pi}$

The type system is defined as a refinement of input/output types (we inherit the first five rules of Figure 2.2 in Section 2.3), and is given by the rules of Figure 3.4. There is a dedicated typing rule for every kind of prefix (free, $\rho\langle b\rangle$, or bound, $\rho(x)$), according to the sort of the involved name. We write $\Gamma(a)$ for the type associated to a in Γ. $T^{\leftrightarrow}$ stands for T where we switch the top-level capability: $(c^s T)^{\leftrightarrow} = \overline{c}^s T$ where $\overline{\mathtt{o}} = \mathtt{i}$, $\overline{\mathtt{i}} = \mathtt{o}$, $\overline{\sharp} = \sharp$. We sometimes write $\Gamma \vdash P, Q$ to mean "$\Gamma \vdash P$ and $\Gamma \vdash Q$".

$$\frac{\Gamma \vdash a : \mathtt{i}^{\mathbf{r}} T \qquad \Gamma, x : T \vdash P}{\Gamma \vdash a(x).P} \qquad \frac{\Gamma \vdash a : \mathtt{o}^{\mathbf{r}} T \qquad \Gamma, x : T^{\leftrightarrow} \vdash P}{\Gamma \vdash \overline{a}(x).P} \qquad \frac{\Gamma \vdash a : \mathtt{o}^{\mathbf{r}} T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}\langle b\rangle.P}$$

$$\frac{\Gamma \vdash a : \mathtt{i}^{\mathbf{e}} T \qquad \Gamma, x : T^{\leftrightarrow} \vdash P}{\Gamma \vdash a(x).P} \qquad \frac{\Gamma \vdash a : \mathtt{o}^{\mathbf{e}} T \qquad \Gamma, x : T \vdash P}{\Gamma \vdash \overline{a}(x).P} \qquad \frac{\Gamma \vdash a : \mathtt{i}^{\mathbf{e}} T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash a\langle b\rangle.P}$$

Figure 3.4: Typing rules for $\overline{\pi}$

The typing rules for $\mathbf{r}$-names impose a constraint on the **r**eceiving side: all inputs on an $\mathbf{r}$-channel should be bound. Note that $\overline{a}(x).P$ and $(\boldsymbol{\nu} x)\overline{a}x.P$ are *not* equivalent from the point of view of typing: typing a bound output on an $\mathbf{r}$-channel (a) imposes that the transmitted name (x) is used according to the "dual constraint" w.r.t. what a's type specifies: this is enforced using $T^{\leftrightarrow}$ (while names received on a are used according to T). Symmetrical considerations can be made for $\mathbf{e}$-names, that impose constraints on the **e**mitting side.

Remark 3.3.2. The rules for $\mathbf{r}$-channels are exactly those of the π-calculus with i/o-types (π^{io}), and typability of $\mathbf{e}$-free processes coincides with typability in π^{io}.

Remark 3.3.3 ("Double contract")**.** We could adopt a more liberal typing for bound outputs on $\mathbf{r}$ names, and use the rule

$$\frac{\Gamma \vdash a : \mathtt{o}^{\mathbf{r}} T \qquad \Gamma, x : T' \vdash P \qquad T' \leq T}{\Gamma \vdash \overline{a}(x).P}$$

(and its counterpart for inputs on $\mathbf{e}$-names). This would have the effect of typing $\overline{a}(x).P$ like $(\boldsymbol{\nu} x)\overline{a}x.P$. We instead chose to enforce what we call a *"double contract"*: the same way a receiving process uses the bound

name according to the type specified in the channel that is used for reception, the continuation of a bound output uses the emitted name according to $T^{\leftrightarrow}$, the *symmetrised version* of T. This corresponds to a useful programming idiom in π, where it is common to create a name, transmit one capability on this name and use locally the other, dual capability. This idiom is used e.g. in [Vas09] and in [SW01, Sect. 5.7.3]. This choice moreover makes the proofs in Section 3.3.3 easier.

Observe that when a typable process reduces according to

$$\overline{a}(x).P \mid a(x).Q \to (\boldsymbol{\nu} x)(P \mid Q) \ ,$$

if a has type, say, $\sharp^{\mathbf{r}}(\mathtt{o}^s T)$, then in the right hand side process, name x is given type $\sharp^s T$, and the $\sharp$ capability is "split" into $\mathtt{i}^s T$ (used by P) and $\mathtt{o}^s T$ (used by Q)—it would be the other way around if a's sort were $\mathbf{e}$.

We prove subject reduction and define typed barbed congruence in the same way it has be done for the π-calculus.

Lemma 3.3.4 (Properties of typing)**.**

1. *(Weakening) If* $\Gamma \vdash P$ *then* $\Gamma, a : T \vdash P$.
2. *(Strengthening) If* $\Gamma, a : T \vdash P$ *and* $a \notin fn(P)$ *then* $\Gamma \vdash P$.
3. *(Narrowing) if* $T_1 \leq T_2$ *and* $\Gamma, a : T_2 \vdash P$ *then* $\Gamma, a : T_1 \vdash P$*;*
4. *(Substitution) If* $\Gamma, x : T \vdash P$ *and* $\Gamma \vdash b : T$ *then* $\Gamma \vdash P[b/x]$.

Proof. (1) and (2) are straightforward by induction on the typing derivation of P.

We prove (3) by induction, and rely on transitivity of $\leq$ in the base cases (i.e. the only place where Γ is invoked). In cases for which new types are introduced (rules for $\boldsymbol{\nu}$ or bound prefixes), we keep the same types.

(4) follows from (3), instantiating T_1 with $\Gamma(b)$ and T_2 with T, and then from (2). □

Proposition 3.3.5 (Subject reduction)**.** *If* $\Gamma \vdash P$ *and* $P \to Q$ *then* $\Gamma \vdash Q$.

Proof. By induction on the derivation of $P \to Q$. Typing is closed under contexts and structural congruence, since $\equiv$ does not interfere with prefixes. This leaves only the base cases:

- A free prefix synchronises with a bound prefix:

  $$\Gamma \vdash \rho b.P \mid \overline{\rho}(x).Q \to P \mid Q[b/x] \ .$$

  Then we have either $\rho = a$ and $\Gamma(a) = \sharp^{\mathbf{e}} T$ or $\rho = \overline{a}$ and $\Gamma(a) = \sharp^{\mathbf{r}} T$. In both cases, $\Gamma, x : T \vdash Q$ and $\Gamma \vdash b : T$. By Lemma 3.3.4 (4), $\Gamma \vdash Q[b/x]$

- Two bound prefixes synchronise:

  $$\Gamma \vdash \rho(x).P_0 \mid \overline{\rho}(x).P_1 \to (\boldsymbol{\nu} x)(P_0 \mid P_1) \ .$$

  Then for each $i \in \{0, 1\}$ we have either $\Gamma, x : T \vdash P_i$ or $\Gamma, x : T^{\leftrightarrow} \vdash P_i$. Let $T^\sharp$ be defined as T with the toplevel capability replaced with $\sharp$. Then $T^\sharp \leq T$ and $T^\sharp \leq T^{\leftrightarrow}$. By narrowing (Lemma 3.3.4), $\Gamma, x : T^\sharp \vdash P_0, P_1$ and $\Gamma \vdash (\boldsymbol{\nu} x)(P_0 \mid P_1)$.

□

Definition 3.3.6 (Environment extension, typed context)**.** Let Γ, Δ be typing environments. We say that Γ *extends* Δ if the support of Δ is included in the support of Γ, and if $\Delta \vdash x : T$ entails $\Gamma \vdash x : T$ for all x. A context C is a (Γ/Δ)-context, written $\Gamma/\Delta \vdash C$, if C can be typed in the environment Γ, the hole being well-typed in any context that extends Δ.

Compositionality of typing is an easy consequence of the previous definition and of Lemma 3.3.4:

Lemma 3.3.7. *If* $\Delta \vdash P$ *and* $\Gamma/\Delta \vdash C$ *then* $\Gamma \vdash C[P]$.

We now move to the definition of behavioural equivalence. The barb predicate is defined as it is for πI:

Definition 3.3.8 (Barbs)**.** Given $\rho \in \{a, \overline{a}\}$, where a is a name, we say that P exhibits barb ρ, written $P{\downarrow_\rho}$, if $P \mid \overline{\rho}(x).\overline{w} \longrightarrow R \mid \overline{w}$ for some R, where $w \notin \mathrm{fn}(P)$. As usual the *weak barb* $P \Downarrow_\rho$ stands for $P \Rightarrow{\downarrow_\rho}$.

We stick to the notion of typed congruence defined by Pierce and Sangiorgi [PS93]; note that we could also adapt to the typed setting the notion of reduction-closed barbed congruence of Definition 2.1.4.

Definition 3.3.9 (Typed barbed congruence)**.** *Barbed bisimilarity* is the largest symmetric relation $\dot{\approx}$ such that whenever $P \dot{\approx} Q$, $P \downarrow_\rho$ implies $Q \Downarrow_\rho$ and $P \to P'$ implies $Q \Rightarrow \dot{\approx} P'$.

When $\Delta \vdash P, Q$, we say that P and Q are *barbed congruent at* Δ, written $\Delta \rhd P \cong^c Q$, if for all (Γ/Δ)-context C, $C[P] \dot{\approx} C[Q]$.

Duality Symmetry in $\overline{\pi}$ is expressed through a duality operator, similarly to Definition 3.0.1. A result similar to Proposition 3.0.2 can be trivially adapted (Lemma 3.3.11) , but to adapt Proposition 3.0.3 to $\overline{\pi}$, we need more work, as the notion of equivalence in $\overline{\pi}$ is typed.

Definition 3.3.10 (Dual of a process in $\overline{\pi}$)**.** The dual of a process P, written $\overline{P}$, is the process obtained by transforming prefixes as follows:

$$\overline{\overline{a}\langle b\rangle} \triangleq a\langle b\rangle \qquad \overline{a\langle b\rangle} \triangleq \overline{a}\langle b\rangle \qquad \overline{\overline{a}(x)} \triangleq a(x) \qquad \overline{a(x)} \triangleq \overline{a}(x) \ ,$$

and applying dualisation homomorphically to the other constructs.

Lemma 3.3.11 (Duality for reduction)**.** *If* $P \to Q$ *then* $\overline{P} \to \overline{Q}$.

The converse of Lemma 3.3.11 holds, since $\overline{\overline{P}} = P$. Dualising a type means swapping i/o capabilities and $\mathbf{e}/\mathbf{r}$ sorts.

Definition 3.3.12 (Dual of a type)**.** The dual of T, written $\overline{T}$, is defined by setting $\overline{c^s T} \triangleq \overline{c}^{\overline{s}}\,\overline{T}$, with:

$$\overline{\mathbf{r}} = \mathbf{e}, \qquad \overline{\mathbf{e}} = \mathbf{r} \qquad \overline{\mathtt{i}} = \mathtt{o} \qquad \overline{\mathtt{o}} = \mathtt{i} \qquad \overline{\sharp} = \sharp \ .$$

We extend the definition to typing environments, and write $\overline{\Gamma}$ for the dual of Γ.

Lemma 3.3.13 (Duality for typing)**.**

1. *If* $T_1 \leq T_2$ *then* $\overline{T_1} \leq \overline{T_2}$.
2. *If* $\Gamma \vdash P$ *then* $\overline{\Gamma} \vdash \overline{P}$.
3. *If* $\Gamma/\Delta \vdash C$ *then* $\overline{\Gamma}/\overline{\Delta} \vdash \overline{C}$.

Proof. (1): the covariant type operators ($\mathtt{i}^{\mathbf{r}}$ and $\mathtt{o}^{\mathbf{e}}$) are dual of each other, and so are the contravariant operators ($\mathtt{o}^{\mathbf{r}}$ and $\mathtt{i}^{\mathbf{e}}$).

(2) follows from the shape of the typing rules, e.g., the dual of the rule for $\mathtt{i}^{\mathbf{r}}$ is an instance of the rule for $\overline{\mathtt{i}^{\mathbf{r}}} = \mathtt{o}^{\mathbf{e}}$.

(3) holds because if Φ extends Δ then $\overline{\Phi}$ extends $\overline{\Delta}$ (item (1)). □

Most importantly, duality holds for typed barbed congruence. The result is easy in the untyped case, since duality preserves reduction and dualises barbs. On the other hand, we are not aware of the existence of another system having this property in presence of i/o-types (as mentioned before, the statement does not make sense for fusion calculi when equipped with i/o-types: there is no such type system for fusion calculi as we show in Section 3.4).

Theorem 3.3.14 (Duality for $\cong^c$)**.** *If $\Delta \rhd P \cong^c Q$ then $\overline{\Delta} \rhd \overline{P} \cong^c \overline{Q}$.*

Proof. By Lemma 3.3.13, we only have to prove that if $P \mathrel{\dot{\approx}} Q$ then $\overline{P} \mathrel{\dot{\approx}} \overline{Q}$, which follows from the fact that duality preserves reduction (Lemma 3.3.11) and exchanges input barbs with output barbs. □

We reach by this theorem the desired duality property for typed behavioural equivalence. We now move on to the relationship between $\overline{\pi}$ and π, to extensions of both calculi, and finally to applications of $\overline{\pi}$.

3.3.2 A conservative extension

From $\overline{\pi}$ to π^{io}. The π-calculus equipped with i/o-types (that we note π^{io}, as opposed to the untyped version of π) is an asymmetric calculus. In some sense, $\overline{\pi}$ can be seen as a 'dualisation' of π^{io}. This can be formulated rigorously by projecting $\overline{\pi}$ into π^{io}. To define this projection, which we call a *partial dualisation*, we work in an extended version of π^{io}, where capabilities are duplicated: in addition to the $\mathtt{i}, \mathtt{o}, \sharp$ capabilities, we also have capabilities $\underline{\mathtt{i}}, \underline{\mathtt{o}}$ and $\underline{\sharp}$, that intuitively correspond to the image of the "**e**-part" of $\overline{\pi}$ through the encoding. The additional capabilities act exactly like the corresponding usual capabilities (respectively $\mathtt{i}, \mathtt{o}, \sharp$), in particular w.r.t. subtyping and duality. We write π_2^{io} for the resulting calculus. We discuss below (Remark 3.3.21) to what extent the addition of these additional capabilities is necessary. We also rely on π_2^{io} to prove that $\overline{\pi}$ is a conservative extension of the π-calculus in Theorem 3.3.24—π_2^{io} is actually close, operationally, to both calculi.

Definition 3.3.15 (Partial dualisation)**.** We define a translation from typed processes in $\overline{\pi}$ to π_2^{io}. The translation acts on typing derivations: given a derivation δ of $\Gamma \vdash P$ (written $\delta :: \Gamma_\delta \vdash P$), we define a π_2^{io} process noted $[P]^\delta$ as follows:

$$\begin{array}{rcll}
[\rho b.P]^\delta & = & \overline{\rho} b.[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{e}}T \\
[\rho b.P]^\delta & = & \rho b.[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{r}}T \\
[\rho(x).P]^\delta & = & \overline{\rho}(x).[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{e}}T \\
[\rho(x).P]^\delta & = & \rho(x).[P]^{\delta'} & \text{if } \Gamma_\delta(\mathrm{n}(\rho)) = c^{\mathbf{r}}T
\end{array}$$

$$[(\boldsymbol{\nu} a)P]^\delta = [P]^{\delta'} \qquad [0]^\delta = 0 \qquad [!P]^\delta = ![P]^{\delta'} \qquad [P \mid Q]^\delta = [P]^{\delta'_1} \mid [Q]^{\delta'_2}$$

In the above definition, δ' is the subderivation of δ, in case there is only one, and δ'_1 and δ'_2 are the obvious subderivations in the case of parallel composition. We extend the definition to types: T^* stands for T where all occurrences of $c^{\mathbf{r}}$ (resp. $c^{\mathbf{e}}$) are replaced with c (resp. $\overline{c}$, the dual of $\underline{c}$). We define accordingly Γ^*.

Remark 3.3.16. This definition does not depend on the choice of T in the case for $(\boldsymbol{\nu} a)P$ since the definition of $[\cdot]^\cdot$ does not depend on capabilities ($\mathtt{i}/\mathtt{o}/\sharp$), but only on sorts ($\mathbf{r}/\mathbf{e}$). Hence, the same translation could be defined for a simply typed version of $\overline{\pi}$.

Lemma 3.3.17. *If $\delta :: \Gamma \vdash P$ (in $\overline{\pi}$), then $\Gamma^* \vdash [P]^\delta$ (in π_2^{io}).*

Proof. In moving from Γ to Γ^*, we replace $\mathtt{i}^{\mathbf{e}}$ (resp. $\mathtt{o}^{\mathbf{e}}$, $\mathtt{i}^{\mathbf{r}}$, $\mathtt{o}^{\mathbf{r}}$) with $\underline{\mathtt{o}}$ (resp. $\underline{\mathtt{i}}$, $\mathtt{i}$, $\mathtt{o}$). This transformation preserves the subtyping relation. Moreover, the rules to type prefixes $\mathtt{i}^{\mathbf{r}}, \mathtt{o}^{\mathbf{r}}, \mathtt{i}^{\mathbf{e}}, \mathtt{o}^{\mathbf{e}}$ in $\overline{\pi}$ correspond to the rules for $\mathtt{i}, \mathtt{o}, \underline{\mathtt{o}}, \underline{\mathtt{i}}$ in π_2^{io}: covariant constructors ($\mathtt{i}^{\mathbf{r}}$ and $\mathtt{o}^{\mathbf{e}}$) are mapped to covariant constructors ($\mathtt{i}$ and $\underline{\mathtt{i}}$), same for contravariant constructors ($\mathtt{o}^{\mathbf{r}}$ and $\mathtt{i}^{\mathbf{e}}$ correspond to $\mathtt{o}$ and $\underline{\mathtt{o}}$). □

Lemma 3.3.18. *Whenever $\delta_1 :: \Gamma \vdash P$ and $\delta_2 :: \Gamma \vdash P$ (in $\overline{\pi}$), we have $\Gamma^* \rhd [P]^{\delta_1} \simeq^c [P]^{\delta_2}$ (in π_2^{io}).*

Proof. The relation $\mathcal{R} \triangleq \{([P]^{\delta_1}, [P]^{\delta_2}) \mid \delta_1, \delta_2 :: \Gamma \vdash P\}$ is a strong bisimulation in π and is substitution-closed; hence $\mathcal{R}$ is included in $\simeq^c$, since $[P]^{\delta_i}$ is typable in Γ^* (by Lemma 3.3.17). □

Lemma 3.3.19. *If $\delta_P :: \Gamma \vdash P$ and $\delta_Q :: \Gamma \vdash Q$ then we have the following:*

1. *(P and Q have the same barbs) iff ($[P]^{\delta_P}$ and $[Q]^{\delta_Q}$ have the same barbs)*

2. *if* $P \to P'$ *then* $[P]^{\delta_P} \to [P']^{\delta}$ *for some* $\delta :: \Gamma \vdash P'$.

3. *if* $[P]^{\delta_P} \to P_1$ *then* $P_1 = [P']^{\delta}$ *with* $P \to P'$ *for some* $\delta :: \Gamma \vdash P'$.

4. $P \dot{\approx} Q$ *iff* $[P]^{\delta_P} \dot{\approx} [Q]^{\delta_Q}$.

Proof. For (1) remark that if $\Gamma(a) = c^{\mathbf{r}}T$ then P and $[P]^{\delta_P}$ have the same barbs on a; if $\Gamma(a) = c^{\mathbf{e}}T$, they have dual barbs on a, but in this case so do Q and $[Q]^{\delta_Q}$.

For (2) and (3), we remark that $[\cdot]^{\delta}$ is compositional and preserves the fact that two prefixes can interact—even when moving to a different δ.

(4) is a consequence of (1), (2), (3). □

Proposition 3.3.20 (Full abstraction)**.** *If* $\delta_P :: \Gamma \vdash P$ *and* $\delta_Q :: \Gamma \vdash Q$ *then*

$$\Gamma \rhd P \cong^c Q \text{ (in } \overline{\pi}) \quad \textit{iff} \quad \Gamma^* \rhd [P]^{\delta_P} \cong^c [Q]^{\delta_Q} \text{ (in } \pi_2^{\mathrm{io}}) \ .$$

Proof. Soundness: given a derivation $\gamma :: \Delta/\Gamma \vdash C$, we build $[C]^{\gamma}$ which is a (Δ^*/Γ^*)-context. Then $[C]^{\gamma}[[P]^{\delta_P}] = [C[P]]^{\beta_P}$ for some β_P and we can rely on barbed congruence in π_2^{io} to establish $[C[P]]^{\beta_P} \dot{\approx} [C[Q]]^{\beta_Q}$. By Lemma 3.3.19, we deduce $C[P] \dot{\approx} C[Q]$.

Completeness: we define the reverse translation $\{\cdot\}$- of $[\cdot]$- and reason as above to prove its soundness. Using the fact that $\delta_P :: \Gamma \vdash P$ implies $\{[P]^{\delta_P}\}^{\delta_P^*} = P$ (where $\delta_P^* :: \Gamma^* \vdash [P]^{\delta}$ is the derivation obtained by Lemma 3.3.17) the soundness of $\{\cdot\}$- implies the completeness of $[\cdot]$-, and vice versa. □

Remark 3.3.21 (π_2^{io} vs π^{io})**.** We can make two remarks about the above result.

First, it would seem natural to project directly onto π^{io}, by mapping capabilities $\mathtt{i}^{\mathbf{r}}$ and $\mathtt{o}^{\mathbf{e}}$ into $\mathtt{i}$, and $\mathtt{o}^{\mathbf{r}}$ and $\mathtt{i}^{\mathbf{e}}$ into $\mathtt{o}$. However, the result of Proposition 3.3.20 would not hold in this case. The intuitive reason is that in doing so, we would allow two names having different sorts in $\overline{\pi}$ to be equated in the image of the encoding, thus giving rise to additional observations (since we cannot equate names having different sorts in $\overline{\pi}$). Technically, this question is reminiscent of the problem of closure of bisimilarity under substitutions in the π-calculus.

Second, the key ingredient in the definition of partial dualisation is to preserve the distinction between names having originally different sorts in the $\overline{\pi}$ process. It is possible to define an encoding of π_2^{io} into *a dyadic version* of π^{io} (without the extra capabilities), in order to do so, but we do not enter into specifics here.

Lemma 3.3.22. *Suppose* $\Delta \vdash P, Q$ *holds in* π^{io}*. Then* $\Delta \rhd P \cong^c Q$ *(in* π^{io} *) iff* $\Delta \rhd P \cong^c Q$ *(in* π_2^{io}*).*

Proof. The right-to-left implication is immediate because any π^{io}-context is a π_2^{io}-context. To show the converse, we observe that a (Γ/Δ)-context in π_2^{io} is a (Γ'/Δ)-context in π^{io}, where Γ' is Γ where every $\underline{c}$ capability is replaced with c. □

From π^{io} to $\overline{\pi}$. $\overline{\pi}$ contains π^{io}, the π-calculus with i/o-types: the rules for $\mathbf{r}$-channels are exactly those of π^{io}, and typability of $\mathbf{e}$-free processes coincides with typability in π^{io}. More precisely we can say that $\overline{\pi}$ is a conservative extension of π^{io}. In π^{io} we rely on typed barbed congruence as defined in [SW01], which is essentially the same as $\cong^c$ in $\overline{\pi}$. Before presenting the result, the following remark introduces some notation.

Remark 3.3.23. Suppose $\delta :: \Gamma \vdash P$, in π^{io}. Then $\delta^{\mathbf{r}} :: \Gamma^{\mathbf{r}} \vdash P$ in $\overline{\pi}$, where $\Gamma^{\mathbf{r}}$ stands for Γ in which all types are decorated with $\mathbf{r}$ and $\delta^{\mathbf{r}}$ stands for δ where all usages of the typing rule for restriction introduce an $\mathbf{r}$-type. Moreover $[P]^{\delta^{\mathbf{r}}} = P$ (using the translation from Definition 3.3.15).

Theorem 3.3.24 (Conservative extension)**.** *Suppose* $\Gamma \vdash P, Q$ *holds in* π^{io}*. Then* $\Gamma \rhd P \cong^c Q$ *(in* π^{io}*) iff* $\Gamma^{\mathbf{r}} \rhd P \cong^c Q$ *(in* $\overline{\pi}$*).*

Proof. We use π_2^{io} as an intermediate calculus. By Remark 3.3.23, let δ_P, δ_Q be derivations of $\Gamma^{\mathbf{r}} \vdash P$ and $\Gamma^{\mathbf{r}} \vdash Q$ such that $P = [P]^{\delta_P}$ and $Q = [Q]^{\delta_Q}$. By Proposition 3.3.20, the right hand side is equivalent to $(\Gamma^{\mathbf{r}})^* \rhd [P]^{\delta_P} \cong^c [Q]^{\delta_Q}$ (in π_2^{io}). By hypothesis, and since $(\Gamma^{\mathbf{r}})^* = \Gamma$, the latter is equivalent to $\Gamma \rhd P \cong^c Q$ (in π_2^{io}). Lemma 3.3.22 allows us to finish the proof. □

The result above shows that π can be embedded rather naturally into $\overline{\pi}$. This is in contrast with fusion calculi, where the equivalence on π-calculus terms induced by the embedding into fusions does not coincide with barbed congruence or barbed equivalence in the π-calculus (see Section 3.2).

Remark 3.3.25 ($\overline{\pi}$ and existing symmetric calculi)**.** $\overline{\pi}$ contains the π-calculus, and hence contains (the typed version of) πI, the π-calculus with internal mobility (see [SW01]). On the other hand, because free inputs and free outputs are not allowed to interact in $\overline{\pi}$, $\overline{\pi}$ fails to represent the fusion calculus. This is consistent with Section 3.4, which says that fusion calculi do not have i/o-types, whereas $\overline{\pi}$ does.

3.3.3 Application: relating encodings of the λ-calculus

The core theory of $\overline{\pi}$ now established, we use it to make a correspondence between two encodings of the λ-calculus in the π-calculus: one from Van Bakel and Vigliotti [vBV09], and the original encoding of Milner. Now, the former actually encode *strong* call-by-name—reductions may also take place inside a λ-abstraction. We therefore compare van Bakel and Vigliotti's encoding with the strong variant of Milner's encoding, obtained by replacing an input with a delayed input, following the work of Merro [Mer00] (in a delayed input $a(x)\!:\!P$, the continuation P may perform transitions not involving the binder x even when the head input at a has not been consumed).

For this, we develop an extension of the core theory of $\overline{\pi}$ to incorporate delayed input to prove that the two encodings are the dual of one another. This is achieved by first embedding the π-terms of the λ-encodings into $\overline{\pi}$, and then applying behavioural laws of $\overline{\pi}$. The correctness of these transformations is justified using i/o-types (essentially to express the conditions under which a link can be erased in favour of a substitution). Some of the transformations needed for the λ-encodings, however, are proved only for barbed bisimilarity; see the remarks concluding the section for a discussion.

Extending $\overline{\pi}$ Based on $\overline{\pi}$, we develop an extension, called $\overline{\pi}^{\mathbf{a}}$, with forms of asynchronous communication and polyadicity. The extension to polyadic communication is standard. Asynchronous communication is added via the inclusion of *delayed* prefixes: $a(x)\!:\!P$ (resp. $\overline{a}(x)\!:\!P$) stands for a *(bound) delayed input (resp. output)* prefix. The intuition behind delayed prefixes is that they allow the continuation of the prefix to interact, as long as the performed action is not causally dependent on the prefix itself—this is made more precise below. Intuitively asynchrony [HT91, ACS96] is useful when reasoning about encodings of the λ-calculus because in a β-reduction $(\lambda x.M)N \to M[N/x]$ the "output" part N has no continuation. It is also useful to have asynchrony in input because the considered λ-strategy allows reduction under a λ-abstraction. Moreover asynchrony allows us to derive some transformation laws involving link processes (Section 3.3.3). Note that synchronous *bound* prefixes are still necessary, to encode the argument of an application.

Delayed prefixes are typed like bound input and output prefixes in Section 3.3.1. Types are refined with two new sorts that enforce asynchrony: $\mathbf{d}$ to force inputs to be bound and delayed, $\mathbf{a}$ to force outputs to be bound and delayed—we call such outputs *asynchronous*. For instance, if we have $a : \sharp^{\mathbf{r}}_{\mathbf{d}} T$ for some T, then all inputs at a are bound and delayed. We also include recursive types.

$$T ::= c_t\langle {}^{s_1}T_1, \ldots, {}^{s_n}T_n\rangle \mid \mathbf{1} \mid \mu X.T \mid X \qquad\qquad s ::= \mathbf{e} \mid \mathbf{r} \qquad\qquad t ::= \mathbf{d} \mid \mathbf{a}$$

In the polyadic case, $\mathbf{e}/\mathbf{r}$ sorts are given to each element of the transmitted tuple. We present here only the typing rule for delayed input, in polyadic form, to illustrate how we extend the type system of Section 3.3.1.

$$\frac{\Gamma \vdash a : \mathtt{i}_t\langle {}^{s_1}T_1, \ldots {}^{s_n}T_n\rangle \quad \Gamma, x_1 : T_1^{s_1}, \ldots, x_n : T_n^{s_n} \vdash P}{\Gamma \vdash a(x_1, \ldots, x_n)\!:\!P}$$

(with, when applying this typing rule, $T^{\mathbf{r}} = T$ and $T^{\mathbf{e}} = T^{\leftrightarrow}$). The sort $\mathbf{d}$ (resp. $\mathbf{a}$) is forbidden in the rules to type non-delayed input (resp. output) prefixes.

The definition of operational semantics is extended as follows to handle delayed prefixes (below, $\overline{\rho}(y)P$ stands for either $\overline{\rho}(y).P$ or $\overline{\rho}(y){:}P$):

$$\begin{array}{rcll} P \mid \rho(x){:}Q & \equiv & \rho(x){:}(P \mid Q) & \text{if } x \notin \mathrm{fn}(P) \\ \rho_1(y){:}\rho_2(x){:}P & \equiv & \rho_2(x){:}\rho_1(y){:}P & \text{if } \mathrm{n}(\rho_1) \neq x, x \neq y, y \neq \mathrm{n}(\rho_2) \\ (\boldsymbol{\nu} y)\rho(x){:}P & \equiv & \rho(x){:}(\boldsymbol{\nu} y)P & \text{if } x \neq y, y \neq \mathrm{n}(\rho) \end{array}$$

$$\frac{}{\rho(x){:}(\overline{\rho}(y)P \mid Q) \to (\boldsymbol{\nu} y)(P \mid Q)[y/x]} \qquad \frac{}{\rho(x){:}(\overline{\rho}b.P \mid Q) \to (P \mid Q)[b/x]} \qquad \frac{P \to Q}{\rho(x){:}P \to \rho(x){:}Q}$$

Barbs are defined as in Section 3.3.1, with an additional clause saying that if ρ is a barb of P and $\mathrm{n}(\rho) \neq x$, then ρ is a barb of $\rho'(x){:}P$ for any ρ'.

The results of Section 3.3.1 hold for this extended calculus, with similar proofs:

Proposition 3.3.26 (Duality in $\overline{\pi}^{\mathbf{a}}$)**.**

1. *Duality of typing:* $\Gamma \vdash P \Rightarrow \overline{\Gamma} \vdash \overline{P}$.
2. *Duality of barbed congruence:* $\Gamma \rhd P \cong^c Q \Rightarrow \overline{\Gamma} \rhd \overline{P} \cong^c \overline{Q}$.

The counterpart of Theorem 3.3.24 also holds in $\overline{\pi}^{\mathbf{a}}$, where types also specify how names have to be used in delayed prefixes. It can be stated w.r.t. $\pi^{\mathrm{io},\mathbf{a}}$, which is defined as π^{io} with additional typing information to specify which names have to be used asynchronously.

The extensions $\overline{\pi}^{\mathbf{a}}$ and $\pi^{\mathrm{io},\mathbf{a}}$ are asynchronous versions of $\overline{\pi}$ and π^{io} in the sense that interaction is no longer a synchronous handshaking between two processes: for at least one of the processes, the occurrence of the interaction is not observable because the consumed action is not blocking for a continuation.

Theorem 3.3.27 (Conservative extension, $\overline{\pi}^{\mathbf{a}}$)**.** *Suppose we have* $\Gamma \vdash P$ *and* $\Gamma \vdash Q$ *in* $\pi^{\mathrm{io},\mathbf{a}}$*. Then* $\Gamma \rhd P \cong^c Q$ *(in* $\pi^{\mathrm{io},\mathbf{a}}$*) iff* $\Gamma^{\mathbf{r}} \rhd P \cong^c Q$ *(in* $\overline{\pi}^{\mathbf{a}}$*)* .

Reasoning about Links, a transformation from $\mathtt{o}^{\mathbf{e}}_{\mathbf{a}}$ to $\mathtt{i}^{\mathbf{r}}_{\mathbf{a}}$ We now move to the development of a technical lemma about the validity of a transformation which is used for the analysis of λ-calculus encodings. Differently from partial dualisation (Definition 3.3.15), the transformation, written $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$, modifies prefixes, beyond simple dualisation, by introducing link processes. It also acts on types, by mapping $\mathbf{e}$-names onto $\mathbf{r}$-names.

Definition 3.3.28. We set $\langle\!\langle ab.P \rangle\!\rangle^{\mathbf{er}} = a(x).(x \twoheadrightarrow b \mid \langle\!\langle P \rangle\!\rangle^{\mathbf{er}})$, where $x \twoheadrightarrow b = !x(z).\overline{b}z$ is called a *link process*. We also define $\langle\!\langle \rho(x).P \rangle\!\rangle^{\mathbf{er}} = \rho(x).\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ and similarly for delayed prefixes. $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ leaves free outputs unchanged and acts homomorphically on the other constructors.

The transformation $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ removes all free inputs and inserts free outputs (in the link process). We therefore expect it to return plain π processes. Moreover, the process computed in the translation of free input behaves as expected provided only the *input capability* is transmitted (the link process *at the receiver's side* exerts the input capability on x). Accordingly, we define $T_{\mathtt{oe}} = \mu X.\mathtt{o}^{\mathbf{e}}_{\mathbf{a}}X = \mathtt{o}^{\mathbf{e}}_{\mathbf{a}}\mathtt{o}^{\mathbf{e}}_{\mathbf{a}}\mathtt{o}^{\mathbf{e}}_{\mathbf{a}}\cdots$, and $T_{\mathtt{ir}} = \mu X.\mathtt{i}^{\mathbf{r}}_{\mathbf{a}}X = \mathtt{i}^{\mathbf{r}}_{\mathbf{a}}\mathtt{i}^{\mathbf{r}}_{\mathbf{a}}\mathtt{i}^{\mathbf{r}}_{\mathbf{a}}\cdots$. We let $\Gamma_{\mathtt{ir}}$ (resp. $\Gamma_{\mathtt{oe}}$) range over environments mapping all names to some $c^{\mathbf{r}}_{\mathbf{a}}T_{\mathtt{ir}}$ (resp. $c^{\mathbf{e}}_{\mathbf{a}}T_{\mathtt{oe}}$), for $c \in \{\mathtt{i}, \mathtt{o}, \sharp\}$.

Lemma 3.3.29 (Typing for $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$)**.** *If* $\Gamma_{\mathtt{oe}} \vdash P$ *then* $\Gamma_{\mathtt{ir}} \vdash \langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ *for some* $\Gamma_{\mathtt{ir}}$.

Proof. We prove by induction on P that if $\Gamma \vdash P$ then $\overline{\Gamma} \vdash \langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$. In the case for $\boldsymbol{\nu}$ we always introduce the type $\sharp^{\mathbf{r}}_{\mathbf{a}}T_{\mathtt{ir}}$. For bound prefixes we replace $c^{\mathbf{e}}_{\mathbf{a}}T_{\mathtt{oe}}$ with $\overline{c}^{\mathbf{r}}_{\mathbf{a}}T_{\mathtt{ir}}$, and for free inputs we type links with $T_{\mathtt{ir}}$ types. □

As this result shows, $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ yields processes that only transmit the input capability. This is reminiscent of the localised π-calculus ([SW01], Chapter 5.6) where only the output capability is passed.

It can be noted that Lemma 3.3.29 holds because we enforce a "double contract" in the typing rules (cf. Remark 3.3.3), which allows us to typecheck bound prefixes as $\mathbf{e}$-names (before the transformation) and as $\mathbf{r}$-names (after).

The relationship between P and $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ is given in terms of barbed expansion precongruence, which is a preorder in between strong and weak barbed congruence.

Definition 3.3.30 (Barbed expansion precongruence)**.** *Barbed expansion* is the largest relation $\dot{\lesssim}$ such that whenever $P \dot{\lesssim} Q$,

- if $P \longrightarrow P'$ then $Q \longrightarrow\!\!\!\!\Longrightarrow Q'$ with $P' \dot{\lesssim} Q'$;
- if $Q \longrightarrow Q'$ then $P \longrightarrow P'$ or $P = P'$ with $P' \dot{\lesssim} Q'$;
- $P \downarrow_\rho$ implies $Q \Downarrow_\rho$, and $Q \downarrow_\rho$ implies $P \downarrow_\rho$.

We call *(resp. typed) barbed expansion precongruence ($\lesssim^c$)* the induced (resp. typed) precongruence.

Lemma 3.3.31 (Properties of links)**.**

1. $a : \mathtt{i}^{\mathbf{r}}_{\mathbf{a}} T_{\mathtt{ir}}, b : \mathtt{o}^{\mathbf{r}}_{\mathbf{a}} T_{\mathtt{ir}} \rhd a \twoheadrightarrow b \lesssim^c (\boldsymbol{\nu} x)(a \twoheadrightarrow x \mid x \twoheadrightarrow b)$.
2. *If* $\Gamma_{\mathtt{oe}}, a : T_{\mathtt{oe}} \vdash P$ *then* $\Gamma_{\mathtt{ir}} \rhd \langle\!\langle P \rangle\!\rangle^{\mathbf{er}}[b/a] \lesssim^c (\boldsymbol{\nu} a)(a \twoheadrightarrow b \mid \langle\!\langle P \rangle\!\rangle^{\mathbf{er}})$.

Proof. 1. The law is valid for the ordinary π-calculus (and is substitution-closed); Lemma 3.3.27 transfers the result to $\overline{\pi}$.

2. By typing, a free output involving a in $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ is necessarily in a link; in this case, we can use (1). The other kind of interaction is with some $\overline{a}(x){:}Q$ in $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$, and $\overline{b}(x){:}Q[b/a]$ behaves like $(\boldsymbol{\nu} a)(a \twoheadrightarrow b \mid \overline{a}(x){:}Q[b/a])$. □

We use Lemma 3.3.31 to deduce operational correspondence for the encoding.

Lemma 3.3.32 (Operational correspondence)**.** *Suppose that* $\Gamma_{\mathtt{oe}} \vdash P$.

1. $P \downarrow \rho$ *iff* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \downarrow \rho$.
2. *If* $P \to P'$ *then* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \to \gtrsim^c \langle\!\langle P' \rangle\!\rangle^{\mathbf{er}}$.
3. *If* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \to P_1$ *then* $P \to P'$ *and* $P_1 \gtrsim^c \langle\!\langle P' \rangle\!\rangle^{\mathbf{er}}$ *for some* P'.

A version of these results in the weak case can also be proved, for barbed expansion. Notably, P and $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}$ exhibit the same weak barbs.

Lemma 3.3.33. *If* $\Gamma_{\mathtt{oe}} \vdash P, Q$ *then* $P \dot{\approx} Q$ *iff* $\langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \dot{\approx} \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}$.

Proof sketch. We show that $\dot{\gtrsim}\{(\langle\!\langle P \rangle\!\rangle^{\mathbf{er}}, \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}) \mid P \dot{\approx} Q\}\dot{\lesssim}$ and $\{(P, Q) \mid \langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \dot{\approx} \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}\}$ are weak barbed bisimulations. We then use the adaptation of Lemma 3.3.32 to the weak case, for barbed expansion. □

Lemma 3.3.34. *If* $\Gamma_{\mathtt{oe}} \vdash P, Q$ *and* $\Gamma_{\mathtt{ir}} \rhd \langle\!\langle P \rangle\!\rangle^{\mathbf{er}} \cong^c \langle\!\langle Q \rangle\!\rangle^{\mathbf{er}}$ *then* $\Gamma_{\mathtt{oe}} \rhd P \cong^c Q$.

Proof sketch. We need to apply the transformation only to names typed with $T_{\mathtt{ir}}$-types, inside contexts that might not be typed in the same way. In order to do so, we define a type system with marks on types, such that only $T_{\mathtt{ir}}$-types are marked and such that the marking propagates onto the names of the typed processes.

We modify the encoding $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$ to only operate on those marked prefixes. For every $(\Delta/\Gamma_{\mathtt{oe}})$-context C, its encoding $\langle\!\langle C \rangle\!\rangle^{\mathbf{er}}$ is a $(\Delta'/\Gamma_{\mathtt{ir}})$-context. Thanks to the compositionality of $\langle\!\langle \cdot \rangle\!\rangle^{\mathbf{er}}$, the hypothesis of the lemma implies the equivalence $\langle\!\langle C[P] \rangle\!\rangle^{\mathbf{er}} \dot{\approx} \langle\!\langle C[Q] \rangle\!\rangle^{\mathbf{er}}$. We then adapt the proof of Lemma 3.3.33 to this marked encoding. □

An analysis of van Bakel and Vigliotti's encoding Using the technical tools we introduced, we can move on to the study of the lambda encodings. We start from an adaptation of Milner's call-by-name (cbn) encoding of [Mil90] to *strong* cbn, which also allows reductions to occur under λ-abstractions. We obtain this by using a delayed prefix in the clause for λ-abstraction. The encoding, noted $[\![\cdot]\!]^{\mathcal{M}}$, is defined as follows:

$$[\![x]\!]^{\mathcal{M}}_p \triangleq \overline{x}p$$

$$[\![\lambda x.M]\!]^{\mathcal{M}}_p \triangleq p(x,q).[\![M]\!]^{\mathcal{M}}_q$$

$$[\![MN]\!]^{\mathcal{M}}_p \triangleq (\boldsymbol{\nu} q)\big([\![M]\!]^{\mathcal{M}}_q \mid (\boldsymbol{\nu} x)(\overline{q}\langle x,p\rangle \mid !x(r).[\![N]\!]^{\mathcal{M}}_r)\big)$$

The other encoding we analyse, taken from [vBV09], is written $[\![\cdot]\!]^{\mathcal{B}}$:

$$[\![x]\!]^{\mathcal{B}}_p \triangleq x(p').p' \rightarrowtriangle p$$

$$[\![\lambda x.M]\!]^{\mathcal{B}}_p \triangleq \overline{p}(x,q).[\![M]\!]^{\mathcal{B}}_q$$

$$[\![MN]\!]^{\mathcal{B}}_p \triangleq (\boldsymbol{\nu} q)\big([\![M]\!]^{\mathcal{B}}_q \mid q(x,p').(p' \rightarrowtriangle p \mid !\overline{x}(r).[\![N]\!]^{\mathcal{B}}_r)\big)$$

Note that $[\![\cdot]\!]^{\mathcal{B}}$ is written in [vBV09] using asynchronous free output and restriction instead of delayed bound output. We can adopt a more concise notation since $(\boldsymbol{\nu} x)(\overline{a}x \mid P)$ and $\overline{a}(x).P$ are strongly bisimilar processes, and similarly for $x(p').p' \rightarrowtriangle p$ and $x(p').p' \rightarrowtriangle p$. (Another difference is that the replication in the encoding of the application is guarded, as in [vBV10], to force a tighter operational correspondence between reductions in λ and in the encodings.)

As remarked above, $[\![\cdot]\!]^{\mathcal{B}}$ and $[\![\cdot]\!]^{\mathcal{M}}$ differ considerably because they engage in quite different dialogues with their environments: in $[\![\cdot]\!]^{\mathcal{M}}$ a function receives its argument via an input, in $[\![\cdot]\!]^{\mathcal{B}}$ it interacts via an output. Differences are also visible in the encodings of variables and application (e.g. the use of links).

To compare the encodings $[\![\cdot]\!]^{\mathcal{M}}$ and $[\![\cdot]\!]^{\mathcal{B}}$, we introduce an intermediate encoding, noted $[\![\cdot]\!]^{\mathcal{I}}$, which is defined as the dual of $[\![\cdot]\!]^{\mathcal{M}}$ (in $\overline{\pi}$):

$$[\![x]\!]^{\mathcal{I}}_p \triangleq xp$$

$$[\![\lambda x.M]\!]^{\mathcal{I}}_p \triangleq \overline{p}(x,q).[\![M]\!]^{\mathcal{I}}_q$$

$$[\![MN]\!]^{\mathcal{I}}_p \triangleq (\boldsymbol{\nu} q)\big([\![M]\!]^{\mathcal{I}}_q \mid (\boldsymbol{\nu} x)(q\langle x,p\rangle \mid !\overline{x}(r).[\![N]\!]^{\mathcal{I}}_r)\big)$$

Note that while $[\![\cdot]\!]^{\mathcal{M}}$ and $[\![\cdot]\!]^{\mathcal{B}}$ can be expressed in π, $[\![\cdot]\!]^{\mathcal{I}}$ uses free input, and does thus not define π-calculus processes.

The three encodings given above are based on a similar usage of names. Two kinds of names are used: we refer to names that represent continuations (p, p', q, r in the encodings) as *handles*, and to names that stand for λ-calculus parameters (x, y, z) as *λ-variables*. Here is how these encodings can be typed in $\overline{\pi}$:

Lemma 3.3.35 (Typing the encodings)**.** *$[\![\cdot]\!]^{\mathcal{M}}$, $[\![\cdot]\!]^{\mathcal{B}}$ and $[\![\cdot]\!]^{\mathcal{I}}$ yield processes which are typable with the respective typing environments $\Gamma_{\mathcal{M}}, \Gamma_{\mathcal{B}}, \Gamma_{\mathcal{I}}$, where:*

- *$\Gamma_{\mathcal{M}}$ types λ-variables with $\mathtt{o}^{\mathtt{r}}_{\mathtt{a}}H$ and handles with $H = \mu X.\mathtt{i}^{\mathtt{r}}_{\mathtt{d}}\langle \mathtt{o}^{\mathtt{r}}_{\mathtt{a}}X, X\rangle$;*
- *$\Gamma_{\mathcal{B}}$ uses respectively $\mathtt{i}^{\mathtt{r}}_{\mathtt{d}}G$ and $\mathtt{i}^{\mathtt{r}}_{\mathtt{a}}\langle \mathtt{o}^{\mathtt{r}}_{\mathtt{d}}G, G\rangle$ where $G = \mu Y.\mathtt{i}^{\mathtt{r}}_{\mathtt{a}}\langle \mathtt{o}^{\mathtt{r}}_{\mathtt{d}}Y, Y\rangle$;*
- *$\Gamma_{\mathcal{I}}$ is the dual of $\Gamma_{\mathcal{M}}$ (that is, it uses $\mathtt{i}^{\mathtt{e}}_{\mathtt{d}}\overline{H}$ and $\overline{H} = \mu Z.\mathtt{o}^{\mathtt{e}}_{\mathtt{a}}\langle \mathtt{i}^{\mathtt{e}}_{\mathtt{d}}Z, Z\rangle$).*

Encoding $[\![\cdot]\!]^{\mathcal{I}}$ can be obtained from $[\![\cdot]\!]^{\mathcal{M}}$ by duality. The only difference between $[\![\cdot]\!]^{\mathcal{I}}$ and $[\![\cdot]\!]^{\mathcal{B}}$ is the presence of two links. We rely on a link transformation similar to $\langle\!\langle\cdot\rangle\!\rangle^{\mathtt{er}}$ to move from $[\![\cdot]\!]^{\mathcal{I}}$ to $[\![\cdot]\!]^{\mathcal{B}}$. Thus, by composing the results on duality and on the transformation, we are able to go from $[\![\cdot]\!]^{\mathcal{M}}$ to $[\![\cdot]\!]^{\mathcal{B}}$.

Proposition 3.3.36. *Given two λ-terms M and N, we have $[\![M]\!]^{\mathcal{M}}_p \mathrel{\dot{\approx}} [\![N]\!]^{\mathcal{M}}_p$ if and only if $[\![M]\!]^{\mathcal{B}}_p \mathrel{\dot{\approx}} [\![N]\!]^{\mathcal{B}}_p$ (both equivalences are stated in $\pi^{\mathrm{io},\mathtt{a}}$).*

Proof sketch. By duality, $[\![M]\!]_p^{\mathcal{M}} \dot{\approx} [\![N]\!]_p^{\mathcal{M}}$ iff $[\![M]\!]_p^{\mathcal{I}} \approx [\![N]\!]_p^{\mathcal{I}}$. To establish that this is equivalent to $[\![M]\!]_p^{\mathcal{B}} \dot{\approx} [\![N]\!]_p^{\mathcal{B}}$, we rely on an adaptation of Lemma 3.3.33. For this, we define a transformation that exploits the ideas presented in the link transformation. In particular, handles (p, p', q, r) are treated like in Definition 3.3.28. The handling of λ-variables (x, y, z) is somehow orthogonal, and raises no major difficulty, because such names are always transmitted as bound (fresh) names. □

Remark 3.3.37 (Call by name)**.** To forbid reductions under λ-abstractions, we could adopt Milner's original encoding, and use an input prefix instead of delayed input in the translation of abstractions. Symmetrically, adapting Van Bakel and Vigliotti's encoding to this strategy would mean introducing a free input prefix—which is natural in $\overline{\pi}$, but is impossible in the π-calculus.

One would naturally want to strengthen the full abstraction in Lemma 3.3.33 from barbed bisimilarity to barbed congruence, which would allow us to replace barbed bisimilarity with typed barbed congruence in Proposition 3.3.36 as well (using the type environments of Lemma 3.3.35). While we believe the result to be true, the proof appears difficult because the link transformation modifies both processes and types, so that the types needed for barbed congruence in the two encodings are different. Therefore also the sets of contexts to be taken into account are different. The problem could be tackled by combining the theory on delayed input and the link bisimilarity in [MS04], and adapting it to a typed setting: currently we only know how to reason this way when the calculus is asynchronous and localised [MS04], which only corresponds to a small fragment of the typed calculus.

Conclusion: on symmetry in $\overline{\pi}$ The calculus $\overline{\pi}$ enjoys properties of duality while being "large", in the sense that it incorporates many of the forms of prefix found in dialects of the π-calculus (free input, bound input, and, in the extension in Section 3.3.3, also delayed input, plus the analogue for outputs), and a non-trivial type system based on i/o-types. This syntactic abundance makes $\overline{\pi}$ a possibly interesting model in which to study various forms of dualities, especially as $\overline{\pi}$ has a close relationship with the π-calculus with i/o-types (π^{io}). This is exemplified in our study of encodings of the λ-calculus, where we have applied $\overline{\pi}$ and its theory to explain an encoding of cbn λ-calculus by van Bakel and Vigliotti: it can be related, via dualities, to Milner's encoding.

Even if $\overline{\pi}$ is a symmetric typed calculus, the symmetry of types is not completely satisfactory, and comes from the construction of $\overline{\pi}$ itself, which is a saturation of π by duality. We would like to study more precisely how types can interact with existing symmetric calculi that are not constructed through a process of saturation and that are more minimal.

3.4 Types: the limitations of symmetric calculi

3.4.1 Types in $\pi\mathrm{I}$

Type consistency in $\pi\mathrm{I}$ Some properties are what we call *monotone*: if they hold for the π-calculus, they hold for all of its subcalculi as well, in particular they hold for $\pi\mathrm{I}$. Monotone properties typically quantify positively over processes and reductions of π. For example, $P \simeq_{\pi} Q$ implies $P \simeq_{\pi\mathrm{I}} Q$ (using the notation of Definition 2.1.4) since it quantifies over fewer contexts in $\pi\mathrm{I}$ than in π, but $P \not\simeq_{\pi} Q$ does not imply $P \not\simeq_{\pi\mathrm{I}} Q$ since P and Q might be distinguished by π contexts that are not $\pi\mathrm{I}$ contexts.

Typing properties are also good examples of monotone properties: since subject reduction holds in π (Lemma 2.3.2), it also holds in $\pi\mathrm{I}$ as well, for the induced type system. A similar notion of *type consistency* is inherited, too: in $\pi\mathrm{I}$ as well as in π, if $a : \mathtt{i}T \vdash P$ then P cannot do an output on a (Corollary 2.3.3).

In the π-calculus, type consistency is not trivial because this a can be sent as data to different parts of the process, where it will instantiate other variables (for example, in $\overline{c}a.Q \mid c(x).R$, x will be replaced with a in R even if there was no relationship between x and a before). Since we cannot know in advance which are those variables, we need some sort of static guarantee that those variables cannot be used as output. Type systems provide this kind of guarantee.

In πI, though, this notion of type consistency is virtually useless as it is presented. Indeed no free name can be emitted and no variable will be instantiated with a, and hence we can statically decide if a name a will be used in input or output, just by looking up the presence of subterms of the form $a(x).P$ or $\overline{a}(x).P$, respectively. However, instead of merely looking at the top-level capabilities (e.g. the $\mathtt{i}$ in $\mathtt{i}T$), we can build a different notion of *type consistency in depth*, more meaningful in πI, although valid in π as well:

Proposition 3.4.1. *Suppose $x_1 : c_1 c_2 \cdots c_n T \vdash P$ where each c_i can be $\mathtt{i}$, $\mathtt{o}$, or $\sharp$. Then, if w is fresh:*

- *if $c_n = \mathtt{i}$ then $P \mid \overline{x_1}(x_2).\overline{x_2}(x_3). \cdots . \overline{x_{n-1}}(x_n).x_n(y).\overline{w} \not\Downarrow_{\overline{w}}$;*
- *if $c_n = \mathtt{o}$ then $P \mid \overline{x_1}(x_2).\overline{x_2}(x_3). \cdots . \overline{x_{n-1}}(x_n).\overline{x_n}(y).\overline{w} \not\Downarrow_{\overline{w}}$.*

Proof. We treat the first case only, and we prove the property using the π-calculus. It is then enough, since reductions and barbs are preserved by structural congruence and removing restrictions, to prove that $P \mid Q \not\Downarrow_{\overline{w}}$ where Q is the tester process $Q \triangleq \overline{x_1}x_2.\overline{x_2}x_3. \cdots . \overline{x_{n-1}}x_n.x_n(y).\overline{w}$. By narrowing we can suppose $c_1 = \sharp$. We have now $\Gamma \vdash P \mid Q$ with $\Gamma = x_1 : U_1, x_2 : U_2, \ldots, x_n : U_n, w : \mathtt{o}\mathbf{1} \vdash P \mid Q$ where $U_j = c_j \cdots c_n T$. Suppose now $P \mid Q \Rightarrow R_2$ with $R_2 \downarrow_{\overline{w}}$. Since $\overline{w}$ was guarded in $(x_n(y).\overline{w})$ in $P \mid Q$, there must have been a synchronisation on x_n with an output on x_n in a process R_1 such that $P \mid Q \Rightarrow R_1 \Rightarrow R_2$. By subject reduction we know that $\Gamma \vdash R_1$ and since $\Gamma(x_n) = \mathtt{i}T$ we reach a contradiction. □

This proposition mainly applies on the input side of the process P, as the tester process is a sequence of outputs. We could have formulated Proposition 3.4.1 in the slightly less general case where $c_j = \mathtt{i}$ for all $j < n$, and we would not have lost much of its meaning. That said, tracking inputs is the best we can do: we cannot ensure any such notion in depth on the output side. For example it is possible to get $a : \mathtt{oi}T \vdash P$ and $P \mid a(x).x(y).\overline{w} \Downarrow_{\overline{w}}$ at the same time, since P can be the process $(\boldsymbol{\nu} x : \sharp T)(\overline{a}\langle x\rangle.\overline{x}(y).0)$. More generally, type guarantees only apply on input processes; this comes from the fact that the contracts imposed by types are only carried by input processes, combined with the fact that no free name can be transmitted to a third party.

The effect of the duality on types The symmetry of πI is not reflected in the structure of typing judgements. Indeed, looking as the typing rule for the output and the restriction:

$$\frac{\Gamma \vdash a : \mathtt{o}T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}b.P} \qquad\qquad \frac{\Gamma, b : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} b)P} \ ,$$

we can build a rule dedicated to the bound output, to get a typing rule for $\overline{a}(b).P$ as if it were a macro for the π-calculus process $(\boldsymbol{\nu} b)\overline{a}b.P$. The premise for the judgement $\Gamma \vdash \overline{a}(b).P$ would be that for some type U such that $U \leq T$, we have $\Gamma \vdash a : \mathtt{o}T$ and $\Gamma, b : U \vdash P$. This is in fact equivalent, using the contravariance of $\mathtt{o}$, to the following rule, which is seemingly symmetric to the rule for the input prefix:

$$\frac{\Gamma \vdash a : \mathtt{o}T \qquad \Gamma, b : T \vdash P}{\Gamma \vdash \overline{a}(b).P} \ .$$

The resulting type system for πI processes is strictly equivalent to the one for π when applied on πI processes, and has rules for prefixes which look symmetric, except for an important detail: the covariance of $\mathtt{i}$ and the contravariance of $\mathtt{o}$. Those variances are opposite, and since the shape of the rules cannot compensate for that, switching between inputs and outputs will change the direction of the variance each time it shall be used.

As an example, consider the process $P \triangleq a(x).\overline{x} \mid \overline{a}(y).(y \mid \overline{y})$. Process P can be typed with typing context $a : \sharp\mathtt{o}T$ (in the left part $a(x).\overline{x}$, the type $\mathtt{io}T$ can be assigned to a (since $\sharp\mathtt{o}T \leq \mathtt{io}T$) and for the right part $\overline{a}(y).(y \mid \overline{y})$, $\mathtt{o}\sharp T$ can be assigned to a (since $\sharp\mathtt{o}T \leq \mathtt{oo}T \leq \mathtt{o}\sharp T$).). However, $\overline{P}$, the dual process of P, can only be typed with $a : \sharp\sharp S$ (because types that are subtypes of both $\mathtt{oi}T_1$ and $\mathtt{i}\sharp T_2$ must be of the form $\sharp\sharp S$).

$$a : \sharp\mathtt{o}T \vdash a(x).\overline{x} \mid \overline{a}(y).(y \mid \overline{y})$$
$$a : \sharp\sharp T \vdash \overline{a}(x).x \mid a(y).(\overline{y} \mid y) \ .$$

Consider now the dual process $\overline{a}(x).x \mid a(y).(\overline{y} \mid y)$. Both capabilities of the input object y are now used in the continuation of the input, and hence the type of a must be a subtype of $\mathtt{i}\sharp S$ for some S, which means it must be of type $\sharp\sharp S$, which cannot be a dual of $\sharp\mathtt{o}T$ as duality is supposed to be an involution.

Worse, duality loses type information: in general, if $a : \mathtt{o}T \vdash P$, then we know that the dual typing will be of the form $a : \mathtt{i}S \vdash \overline{P}$, but we do not know much more than that: indeed, consider that P_n is an iterated version of the previous example: if $P_n = \overline{a}(x_1).(x_1 \mid \overline{x_1}(x_2).(x_2 \mid \overline{x_2}(x_3).(\cdots.(x_n \mid \overline{x_n})\cdots)))$ then $a : \mathtt{o}T \vdash P_n$ holds for any T, but $a : \mathtt{i}S \vdash \overline{P_n}$ holds only for types of the form $S = \sharp\sharp\ldots\sharp U$. Inverting polarities, we also lose information: knowing $a : \mathtt{i}T \vdash Q$ gives you information about how a is used in Q, but for every U, $a : \mathtt{o}U \vdash \overline{Q}$ so we lose the entirety of the type information in input types, except for the top-level constructor.

Conclusion We have seen that in πI, the notion of progress is only meaningful for input processes, and then that duality loses the type information of input types. This amounts to say that no useful type information is preserved when switching from a process P to its dual $\overline{P}$.

As a final remark, these observations about the little relevance of i/o-types in πI are not surprising, given that most names are restricted and that basic capabilities have asymmetric variances. On top of this, one of the central expressiveness results of πI is the encodability of π which goes through several encodings, using as intermediate calculi both the asynchronous π-calculus [HT91, Bou92] and the asynchronous localised π-calculus (ALπ) [Bor98, Mer00]. In fact, ALπ is a calculus where there is no meaningful i/o-typing information (indeed, every non-top-level capability can be replaced with $\mathtt{o}$ in ALπ) and hence the encodings lose all type information by going through ALπ.

3.4.2 Types in fusion calculi

Several points can be made in favour of the study of types in fusion calculi. First, Section 3.3 develops a tool to relate processes of π using duality, with $\overline{\pi}$ as an intermediate calculus. Two ingredients were fundamental, in addition to the symmetry of $\overline{\pi}$: first, $\overline{\pi}$ comes with a natural notion of typed behavioural equivalence; second, $\overline{\pi}$ is a conservative extension of π (Theorem 3.3.24). On the contrary, fusion calculi are not eligible candidates to relate π processes: fusion calculi are extensions of π that are not conservative, since the induced equivalences are different, at least in the untyped case (see equation (3.2)). Moreover, there is no notion of typed equivalence in fusion calculi yet, which naturally leads to the more basic question of types in fusions.

Lastly, another reason to analyse types in these calculi is that the fusion calculus originated from the update calculus by the addition of polyadicity. This addition goes usually [Mil93] together with the addition of simple types. Even if this addition can be done in a rather natural way in the fusion calculi, it suggests the extension of simple types to i/o-types [PS93].

Hence, in this section, we study the addition of types to fusion calculi and more generally to calculi where input is not binding. We highlight a striking difference between the π-calculus and fusion calculi. We present impossibility results for subtyping (and more precisely for general capability-based type systems, implicitly or explicitly involving subtyping) in fusion calculi. In the statement of these results, we assume a few basic properties of type systems for name-passing calculi, such as strengthening, weakening and type soundness, and the validity of the ordinary typing rules for the base operators of parallel composition and restriction.

Calculi with fusions Section 3.2 mentions several fusion calculi. We first identify some properties they all share. For example, when restriction is the only binder (hence the input construct is not binding), we say that the calculus *has a single binder*. If in addition interaction involves fusion between names, so that we have ($\Longrightarrow$ stands for an arbitrary number of reduction steps, and in the right-hand side P and Q can be omitted if they are 0)

$$(\boldsymbol{\nu} c)\,(\overline{a}b.P \mid ac.Q \mid R) \quad \Longrightarrow \quad (P \mid Q \mid R)\{b/c\} \ , \tag{3.3}$$

we say that the calculus *has name-fusions*, or, more briefly, has *fusions*.

We are not requiring that (3.3) is among the rules of the operational semantics of the calculus, just that (3.3) holds. The shape of (3.3) has been chosen so to capture the existing calculi; the presence of R allows us to capture also the calculus of Solos.

In this sense, all single-binder calculi in the literature (update [PV97], chi [Fu97], fusion [PV98a], explicit fusion calculus [GW00], Solos [LV03]) have fusions. In Chapter 4 we will introduce a single-binder calculus without fusions.

Types and subtyping We consider typed versions of languages with fusions. We show that in such languages it is impossible to have a non-trivial subtyping, assuming a few simple and standard typing properties of name-passing calculi.

We consider only basic type systems and basic properties for them. Like in all the calculi that we have seen so far, we assume that processes are only well-typed in type environments, which assign types to names; more sophisticated type systems exist where processes have types too, but our setting does not apply to them. In particular, we assume the standard typing rules for the operators of parallel composition and restriction in name-passing calculi (they are rules T-PAR and T-RES in the π-calculus):

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad\qquad \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} a)\, P} \quad . \tag{3.4}$$

In resource-sensitive type systems, i.e., those for linearity [KPT99, Hüt13] and receptiveness ([SW01], Section 8.2), where one counts certain occurrences of names, the rule for parallel composition has to be modified. As mentioned earlier, we stick to basic type systems, ignoring resource consumption.

We also abstract from previous type systems the basic type construct $\sharp T$, for the type of names carrying values of type T: we assume that the following rule for prefixes $ab.P$ and $\overline{a}b.P$ is *admissible*:

$$\text{T-PREF}\quad \frac{\Gamma(a) = \sharp T \qquad \Gamma(b) = T \qquad \Gamma \vdash P}{\Gamma \vdash \alpha.P}\, \alpha \in \{ab, \overline{a}b\} \tag{3.5}$$

(Here we consider input and output prefixes with a continuation; in calculi in which prefixes may not have a continuation, e.g., the asynchronous π-calculus or Solos, P would be missing from the rules.)

Again, these need not be the typing rules for prefixes; we are just assuming that the rules are valid in the type system. We can observe that the standard rule for prefix would have, as hypotheses,

$$\Gamma \vdash a : \sharp T \qquad \Gamma \vdash b : T \quad .$$

These imply, but are not equivalent to, the hypotheses in (3.5), for instance in presence of subtyping.

As mentioned above, fundamental properties of type systems are:

- Subject Reduction (or Type Soundness): if $\Gamma \vdash P$ and $P \to P'$, then $\Gamma \vdash P'$;
- Weakening: if $\Gamma \vdash P$ and a is fresh, then $\Gamma, a : T \vdash P$;
- Strengthening: whenever $\Gamma, a : T \vdash P$ and a is fresh for P, then $\Gamma \vdash P$;
- Closure under injective substitutions: if $\Gamma, a : T \vdash P$ and b is fresh, then $\Gamma, b : T \vdash P\{b/a\}$.

Definition 3.4.2. A typed calculus with single binder is *plain* if it satisfies Subject Reduction, Weakening, Strengthening, Closure under injective substitutions, and the typing rules (3.4) and (3.5) are admissible.

If the type system admits subtyping, then another fundamental property is narrowing, which authorises, in a typing environment, the specialisation of types:

- Narrowing: if $\Gamma, a : T \vdash P$ and $U \leq T$ then also $\Gamma, a : U \vdash P$.

When narrowing holds, we say that the calculus *supports narrowing*.

A typed calculus *has trivial subtyping* if, whenever $T \leq U$, we have $\Gamma, a : T \vdash P$ iff $\Gamma, a : U \vdash P$. When this is not the case (i.e., there are T, U with $T \leq U$, and T, U are not interchangeable in all typing judgements) we say that the calculus has *meaningful* subtyping.

We are now ready to prove that, under the assumptions of Definition 3.4.2, a calculus with fusions may only have trivial subtyping.

Theorem 3.4.3. *A typed calculus with fusions that is plain and supports narrowing has trivial subtyping.*

Proof. We define the following context:

$$E \triangleq (\boldsymbol{\nu} cb)(\overline{u}b \mid uc \mid \overline{v}a \mid vc \mid [\cdot]) \ .$$

Note that in E we only use b as an output object (in $\overline{u}b$). The intention is that, given some fresh names, u, v, c, and some process P, $E[P]$ should reduce to $P\{a/b\}$. Indeed, by applying hypothesis (3.3) twice, we have

$$\begin{array}{rcl} E[P] = (\boldsymbol{\nu} bc)(\overline{u}b \mid uc \mid \overline{v}a \mid vc \mid P) & \Longrightarrow & (\boldsymbol{\nu} b)(\overline{v}a \mid vb \mid P\{b/c\}) \\ & = & (\boldsymbol{\nu} b)(\overline{v}a \mid vb \mid P) \\ & \Longrightarrow & P\{a/b\} \ . \end{array}$$

Suppose $U \leq T$, we show $\Gamma, a : T \vdash P$ iff $\Gamma, a : U \vdash P$, which means that the type system has trivial subtyping. The implication from left to right is narrowing.

To prove the implication from right to left, suppose $\Gamma, a : U \vdash P$, and prove $\Gamma, a : T \vdash P$. We can pick some name b fresh for P, and deduce, by Closure under injective name substitution, $\Gamma, b : U \vdash P\{b/a\}$.

In the typing environment $\Gamma, b{:}U, u{:}\sharp T, v{:}\sharp T, c{:}T, a{:}T$ the process $\overline{u}b$ is well-typed thanks to Narrowing and Weakening, hence so is $(\overline{u}b \mid uc \mid \overline{v}a \mid vc \mid P\{b/a\})$. By the restriction rule we get $\Gamma, a{:}T, u{:}\sharp T, v{:}\sharp T \vdash E[P\{b/a\}]$, the latter reducing to $P\{b/a\}\{a/b\}$ by (3.6). Since b has been taken fresh, $P\{b/a\}\{a/b\} = P$. Hence, by Subject Reduction, $\Gamma, a{:}T, u{:}\sharp T, v{:}\sharp T \vdash P$. We finally deduce $\Gamma, a : T \vdash P$ by Strengthening. □

One may wonder whether, in Theorem 3.4.3, more limited forms of narrowing, or a narrowing in the opposite direction, would permit some meaningful subtyping. Narrowing is interesting when it is used to modify the type of the values exchanged along a name, that is, the type of the object of a prefix. (In process calculi, communication is the analogous of application for functional languages, and changing the type of a prefix object is similar to changing the type of a function or of its argument.) In other words, disallowing narrowing on objects would make subtyping useless. We now show that allowing *any* form of narrowing, on one prefix object, would force subtyping to be trivial.

Theorem 3.4.4. *We consider a typed calculus with fusions which is plain. We suppose that there is at least one prefix α with object b, where b is different from the subject of α. We further suppose that there are two types S and T such that $S \leq T$ and one of the following forms of narrowing holds for all Γ:*

(1) whenever $\Gamma, b : T \vdash \alpha.0$, we also have $\Gamma, b : S \vdash \alpha.0$;

(2) whenever $\Gamma, b : S \vdash \alpha.0$, we also have $\Gamma, b : T \vdash \alpha.0$.

Then S and T are interchangeable in all typing judgements.

As a consequence, authorising one of the above forms of narrowing for all S and T such that $S \leq T$ implies that the calculus has trivial subtyping.

Proof. For all Δ we prove that $\Delta, x : T \vdash P$ iff $\Delta, x : S \vdash P$. Let x_1, x_2, a_1 and a_2 be fresh names, we define

$$\Delta_i \triangleq \Delta, \ x_i : T, \ x_{3-i} : S \ .$$

We prove that $\Delta_i \vdash P\{x_1/x\}$ implies $\Delta_i \vdash P\{x_2/x\}$ for all $i \in \{1,2\}$. This is enough to conclude, using Weakening, Strengthening and Closure under injective substitutions. We rely on process $D \triangleq \overline{a_1}x_1 \mid \overline{a_2}x_2 \mid a_1y \mid a_2y$ to simulate the substitution of x_1 with x_2:

$$(\boldsymbol{\nu} x_1, y)(D \mid P\{x_1/x\}) \implies P\{x_2/x\} \ . \tag{3.6}$$

This way, it is enough to find some types $T_{a_1}\ T_{a_2}, T_y$ such that $\Delta' \vdash D$, with $\Delta' = \Delta_i, a_1 : T_{a_1}, a_2 : T_{a_2}, y : T_y$. We suppose that the subject of α is either a_1 or a_2, picking the most suitable choice, without loss of generality. We suppose as well that the object b is the most suitable among x_1, x_2 or y.

To illustrate how we choose such types, we provide an example. Suppose: that $i = 1$ (i.e., the type of x_1 is T and the type of x_2 is S), that narrowing of type (1) holds, and that α is an output. Then we take α to be $\overline{a_2}x_2$ and we choose $T_{a_1} = T_{a_2} = \sharp T$ and $T_y = T$. We then have $\Delta' \vdash D$ quite easily: the interesting part is to obtain $a_2 : \sharp T, x_2 : S \vdash \overline{a_2}x_2$ from the composition of the trivial $a_2 : \sharp T, x_2 : T \vdash \overline{a_2}x_2$ and from narrowing of type (1). Then, by Subject Reduction, with (3.6) and Strengthening, we get that $\Delta_1 \vdash P\{x_2/x\}$. This particular case corresponds to the first line of the following table.

The other choices for $T_{a_1}\ T_{a_2}$, and T_y are listed in the table, following three parameters: is i equal to 1 or to 2? Which form of narrowing, between (1) and (2), holds? Is α an output or an input?

i	form	α	T_{a_1}	T_{a_2}	T_y
1	(1)	$\overline{a_2}x_2$	$\sharp T$	$\sharp T$	T
1	(1)	a_1y	$\sharp T$	$\sharp S$	S
1	(2)	$\overline{a_1}x_1$	$\sharp S$	$\sharp S$	S
1	(2)	a_2y	$\sharp T$	$\sharp S$	T
2	(1)	$\overline{a_2}x_2$	$\sharp T$	$\sharp T$	T
2	(1)	a_2y	$\sharp S$	$\sharp T$	S
2	(2)	$\overline{a_1}x_1$	$\sharp S$	$\sharp S$	S
2	(2)	a_1y	$\sharp S$	$\sharp T$	T

Using the hypothesis on α, we can prove in all these cases that $\Delta' \vdash D$, relying on the fact that the type system is plain. The latter hypothesis also gives us $\Delta' \vdash P\{x_1/x\}$. We use rules from (3.4) and Subject Reduction to deduce that $\Delta' \vdash P\{x_2/x\}$. From this, Strengthening is enough to conclude. □

Remark 3.4.5. Theorems 3.4.3 and 3.4.4 apply to all fusion calculi: implicit fusions, explicit fusions, update calculus, chi-calculus, solos.

In calculi of mobile processes, the basis for having subtyping is a type system for *capabilities*, usually the input/output (I/O) capabilities [PS93, HR02]. Another consequence of Theorems 3.4.3 and 3.4.4 is that it is impossible, in plain calculi with fusions, to have an I/O type system; more generally, it is impossible to have any capability-based type system that supports meaningful subtyping.

Actually, to apply the theorems, it is not even necessary for the capability type system to have an explicit notion of subtyping. For Theorem 3.4.3, it is sufficient to have sets of capabilities with a non-trivial ordering under inclusion, meaning that we can find two capability types T and U such that whenever $\Gamma, a : U \vdash P$ holds then also $\Gamma, a : T \vdash P$ holds, but not the converse (e.g., T provides more capabilities than U). We could then impose a subtype relation $\leq$ on types, as the least preorder satisfying $T \leq U$. Theorem 3.4.3 then tells us that type soundness and the other properties of Definition 3.4.2 would require also $U \leq T$ to hold, i.e., T and U are interchangeable in all typing judgements. In other words, the difference between the capabilities in T and U has no consequence on typing. Similarly, to apply Theorem 3.4.4 it is sufficient to find two capability types T and U and a single prefix in whose typing the type U can replace T.

Conclusion The fact that subtyping and the explicit fusion calculus are incompatible is not very surprising, as analysing the possible typing rules for the explicit fusion construct suggests the symmetry constraint on the subtyping relation. It is a little less obvious for the fusion calculus, for which substitutions are more localised, and even less for the update calculus, where the substitutions are directed and lack symmetry with respect to (explicit) fusions.

This impossibility result extends further than just type systems with subtyping, as even capability control involves an implicit notion of subtyping. However, this result does not apply to other kinds of type system, especially those that do not admit a rule of the shape T-PAR (Figure 2.2), for example session types (that can, in fact, be adapted to some fusion calculi [Gue14]) or linear types (we can also recover a linear type system for fusion calculi, although that makes the latter very close to the π-calculus (Section 4.8)).

Chapter 4

Preorders on names

Section 3.4 showed that the symmetry of the relation on names induced by fusions, augmented as explicit fusions appear along the execution of a process, is incompatible with the asymmetry of subtyping. This chapter a study of what happens when we drop the symmetry hypothesis of the induced relation on names, which becomes a preorder relation rather than an equivalence relation. This different relation yields a new calculus, which we write $\pi\mathrm{P}$, for "π with preorders".

In Section 4.1 we describe the calculus $\pi\mathrm{P}$, and in Section 4.2 we show that it recovers the types of the π-calculus. This can be done using a step-by-step semantics (the eager reduction), and then through a somehow more sensible but less compositional semantics, that we see in Section 4.3. Section 4.4 shows that $\pi\mathrm{P}$ features a notion of privacy of names, the same as in the π-calculus, that fusion calculi do not. Section 4.5 develops in turn a compositional semantics, and Section 4.6 gives an axiomatisation of $\pi\mathrm{P}$. Section 4.7 gives an account of the expressiveness of the calculi, with respect to π-calculi and fusion calculi. Section 4.8 presents a syntactical restriction that makes all calculi (π, fusions, and $\pi\mathrm{P}$) equal. Section 4.9 discusses less precise questions about $\pi\mathrm{P}$.

4.1 A π-calculus with preorders, $\pi\mathrm{P}$

We refine the fusion calculi by replacing the equivalence relation on names generated through communication by a preorder, yielding calculus $\pi\mathrm{P}$ ('π with Preorder'). As the preorder on types given by subtyping enables promotions between related types, so the preorder on names of $\pi\mathrm{P}$ enables promotions between related names. Precisely, if a is below a name b in the preorder, then a prefix at a may be promoted to a prefix at b and then interact with another prefix at b. Thus an input $av.P$ may interact with an output $\overline{b}w.Q$; and, if also c is below b, then $av.P$ may as well interact with an output $\overline{c}z.R$.

The ordering on names is introduced by means of the *arc* construct, $a\!\!/b$, that declares the *source* b to be below the *target* a. The remaining operators are the same as for fusion calculi (i.e., those of the π-calculus with bound input replaced by free input). Restriction is the only binder. Note that the calculus will be extended in Sections 4.5 and 4.6 to study its axiomatisation.

$$\pi \;::=\; \overline{a}\langle b\rangle \;\mid\; a\langle b\rangle \qquad\qquad P \;::=\; 0 \;\mid\; P|Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \pi.P \;\mid\; a\!\!/b \;.$$

The semantics of the calculus is given by a reduction relation, which is based on structural congruence (the same as Definition 2.1.1), and is defined as follows (the subscript in $\longrightarrow_{\mathrm{ea}}$, for "eager", will distinguish this from the alternative semantics discussed in Section 4.3):

Definition 4.1.1 (Eager reduction)**.** Eager reduction in $\pi\mathrm{P}$, written $\longrightarrow_{\mathrm{ea}}$, is defined by the following rules:

$$\overline{a/b \mid \overline{b}c.Q \longrightarrow_{\mathrm{ea}} a/b \mid \overline{a}c.Q} \qquad \overline{\overline{a}b.P \mid ac.Q \longrightarrow_{\mathrm{ea}} P \mid Q \mid b/c} \qquad \overline{a/b \mid bc.Q \longrightarrow_{\mathrm{ea}} a/b \mid ac.Q}$$

$$\frac{P \longrightarrow_{\mathrm{ea}} P'}{P \mid R \longrightarrow_{\mathrm{ea}} P' \mid R} \qquad \frac{P \longrightarrow_{\mathrm{ea}} P'}{(\boldsymbol{\nu} a)P \longrightarrow_{\mathrm{ea}} (\boldsymbol{\nu} a)P'} \qquad \frac{P \equiv \longrightarrow_{\mathrm{ea}} \equiv P'}{P \longrightarrow_{\mathrm{ea}} P'}$$

We write $\Longrightarrow_{\mathrm{ea}}$ for the reflexive transitive closure of $\longrightarrow_{\mathrm{ea}}$.

The first rule shows that communication generates an arc. The next two rules show that arcs only act on the subject of prefixes; moreover, they only act on *unguarded* prefixes (i.e., prefixes that are not underneath another prefix). The rules also show that arcs are persistent processes. Acting only on prefix subjects, arcs can be thought of as particles that "redirect prefixes": an arc a/b redirects a prefix at b towards a higher name a.

Arcs remind us of special π-calculus processes, called forwarders or wires [HY95], which under certain hypotheses (the calculus should be asynchronous and localised) allow one to model substitutions; as for arcs, so the effect of forwarders is to replace the subject of prefixes. Note also that arcs are different from explicit substitutions: only the latter are binding, and there can be several arcs acting on the same name.

We present below some examples of reduction. We sometimes omit the object part of prefixes, when it is not important, e.g. $e.P$ can stand for $(\boldsymbol{\nu} x)ex.P$ when $x \notin \mathrm{fn}(P)$.

$$\begin{array}{rr}
 & \overline{a}c.\overline{c}a.e.P \mid ad.de.\overline{a}.Q \\
\longrightarrow_{\mathrm{ea}} & \overline{c}a.e.P \mid de.\overline{a}.Q \mid c/d \\
\longrightarrow_{\mathrm{ea}} & \overline{c}a.e.P \mid ce.\overline{a}.Q \mid c/d \\
\longrightarrow_{\mathrm{ea}} & e.P \mid \overline{a}.Q \mid c/d \mid a/e \\
\longrightarrow_{\mathrm{ea}} & a.P \mid \overline{a}.Q \mid c/d \mid a/e \\
\longrightarrow_{\mathrm{ea}} & P \mid Q \mid c/d \mid a/e
\end{array}$$

Reductions can produce multiple arcs that act on the same name. This may be used to represent certain forms of choice, as in the following processes:

$$\begin{array}{l}
(\boldsymbol{\nu} h, k)\,(bu.cu.\overline{u} \mid \overline{b}h.h.P \mid \overline{c}k.k.Q) \\
\Longrightarrow_{\mathrm{ea}}\ (\boldsymbol{\nu} h, k)\,(\overline{u} \mid h/u \mid k/u \mid h.P \mid k.Q)\ .
\end{array}$$

Arcs h/u and k/u may act on $\overline{u}$, and are therefore in competition with each other. The outcome of the competition determines which process between P and Q is activated. For instance, reduction may continue as follows:

$$\begin{array}{ll}
\longrightarrow_{\mathrm{ea}} & (\boldsymbol{\nu} h, k)\,(\overline{k} \mid h/u \mid k/u \mid h.P \mid k.Q) \\
\longrightarrow_{\mathrm{ea}} & (\boldsymbol{\nu} h, k)\,(h/u \mid k/u \mid h.P \mid Q)\ .
\end{array}$$

The semantics above formalises an *eager* behaviour for arcs: arcs act on prefixes, substituting their subjects, regardless of the possible consequences on future interactions. For instance, in the example above, the arc k/u could have acted on $\overline{u}$ even in the absence of the matching prefix $k.Q$. Under the eager semantics the substitution produced by an arc is a commitment—replacing a with b in a prefix commits that prefix to go along a channel that is at least as high as b in the preorder. In Section 4.3, we formalise a *by-need* behaviour of arcs, in which substitutions are performed only when reduction takes place. In by-need semantics, there is no separate action of commitment.

Definition 4.1.2 (Positive and negative occurrences)**.** In an input $ab.P$ and an arc a/b, the name b has a *negative occurrence*. All other occurrences of names in input, output and arcs are *positive occurrences*. More formally, we define $\mathrm{fn}^+(P)$ and $\mathrm{fn}^-(P)$, the free positive (resp. negative) names of P, as follows:

$$\begin{array}{lll}
\mathrm{fn}^+(\overline{a}b.P) = \{a,b\} \cup \mathrm{fn}^+(P) & \mathrm{fn}^+(ab.P) = \{a\} \cup \mathrm{fn}^+(P) & \mathrm{fn}^+(a/b) = \{a\} \\
\mathrm{fn}^-(\overline{a}b.P) = \mathrm{fn}^-(P) & \mathrm{fn}^-(ab.P) = \{b\} \cup \mathrm{fn}^-(P) & \mathrm{fn}^-(a/b) = \{b\} \\
\mathrm{fn}^-(P \mid Q) = \mathrm{fn}^-(P) \cup \mathrm{fn}^-(Q) & \mathrm{fn}^-((\boldsymbol{\nu} a)P) = \mathrm{fn}^-(P) \setminus \{a\} & \mathrm{fn}^+(0) = \emptyset \\
\mathrm{fn}^+(P \mid Q) = \mathrm{fn}^+(P) \cup \mathrm{fn}^+(Q) & \mathrm{fn}^+((\boldsymbol{\nu} a)P) = \mathrm{fn}^+(P) \setminus \{a\} & \mathrm{fn}^-(0) = \emptyset
\end{array}$$

and we remark that $\mathrm{fn}(P) = \mathrm{fn}^+(P) \cup \mathrm{fn}^-(P)$.

An occurrence in a restriction ($\boldsymbol{\nu} a$) is neither negative nor positive, intuitively because restriction acts only as a binder, and does not stand for an usage of the name (in particular, it does not take part in a substitution).

Negative occurrences are particularly important, as by properly tuning them, different usages of names may be obtained. For instance, a name with zero negative occurrences is a constant (i.e., it is a channel, and may not be substituted); and a name that has a single negative occurrence is like a π-calculus name bound by an input (we investigate this situation in Section 4.7.2). Names having two or more negative occurrences are specific to πP.

The number of negative occurrences of a name is invariant under reduction. Note also that the set of positive occurrences is non-increasing.

Lemma 4.1.3. *If $P \longrightarrow_{\mathrm{ea}} P'$ then for each b, the number of negative occurrences of b in P and P' is the same.*

4.2 Types

We now show how the i/o capability type system from π and its subtyping relation (Section 2.3) can be transplanted to πP. Types in πP are the same as for π:

$$T \;::=\; \mathtt{i}T \;\mid\; \mathtt{o}T \;\mid\; \sharp T \;\mid\; \mathbf{1}$$

Subtyping rules (Figure 4.1) are unchanged as well, they are the same as in the π-calculus (Figure 2.3).

$$\textsc{Sub-refl}\ \frac{}{T \leq T} \qquad \textsc{Sub-trans}\ \frac{T_1 \leq T_2 \qquad T_2 \leq T_3}{T_1 \leq T_3} \qquad \textsc{Sub-}\sharp\mathtt{i}\ \frac{}{\sharp T \leq \mathtt{i}T} \qquad \textsc{Sub-}\sharp\mathtt{o}\ \frac{}{\sharp T \leq \mathtt{o}T}$$

$$\textsc{Sub-ii}\ \frac{T_1 \leq T_2}{\mathtt{i}T_1 \leq \mathtt{i}T_2} \qquad \textsc{Sub-oo}\ \frac{T_1 \leq T_2}{\mathtt{o}T_2 \leq \mathtt{o}T_1}$$

Figure 4.1: Subtyping for i/o-types in πP

The type system for πP is presented in Figure 4.2. With respect to the π-calculus, only the rule for input needs an adjustment, as πP uses free, rather than bound, input. The idea in rule T-InpFree of πP is however the same as in rule T-Inp of π: we look up the type of the object of the prefix, say T, and we require $\mathtt{i}T$ as the type for the subject of the prefix. To understand the typing of an arc a/b, recall that such an arc allows one to replace b with a. Rule T-Arc essentially ensures that a has at least as many capabilities as b, in line with the intuition for subtyping in capability type systems.

$$\textsc{T-Name}\ \frac{\Gamma(a) \leq T}{\Gamma \vdash a : T} \qquad \textsc{T-Nil}\ \frac{}{\Gamma \vdash 0} \qquad \textsc{T-Par}\ \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \textsc{T-Res}\ \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} a)P}$$

$$\textsc{T-Out}\ \frac{\Gamma \vdash a : \mathtt{o}T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}b.P} \qquad \textsc{T-InpFree}\ \frac{\Gamma \vdash a : \mathtt{i}\Gamma(b) \qquad \Gamma \vdash P}{\Gamma \vdash ab.P} \qquad \textsc{T-Arc}\ \frac{\Gamma \vdash a : \Gamma(b)}{\Gamma \vdash a/b}$$

Figure 4.2: i/o-types in the πP-calculus

Common to all premises of T-Inp, T-InpFree and T-Arc is the look up of the type of names that occur negatively (the source of an arc and the object of an input prefix): the type that appears for b in the hypothesis

is *precisely* the type found in the conclusion (within the process or in Γ). In contrast, the types for positive occurrences may be different (e.g., because of subsumption, $\Gamma \vdash a : \mathtt{i}T$ may hold even if $\Gamma(a) \neq \mathtt{i}T$).

Note moreover that we cannot type inputs like outputs: consider the rule

$$\text{T-InpFree2-Wrong}\quad \frac{\Gamma \vdash a : \mathtt{i}T \qquad \Gamma \vdash b : T \qquad \Gamma \vdash P}{\Gamma \vdash ab.P}\ .$$

Rule T-InpFree2-Wrong would accept, for instance, an input ab in an environment Γ where $a : \mathtt{ii}\mathbf{1}$ and $b : \sharp\mathbf{1}$. Using subtyping and subsumption, we could then derive $\Gamma \vdash b\mathtt{i}\mathbf{1}$. In contrast, rule T-InpFree, following the input rule of the π-calculus, makes sure that the object of the input does not have too many capabilities with respect to what is expected in the type of the subject of the input. Enforcing this constraint is necessary for subject reduction. As a counterexample, consider $\Gamma = a : \sharp\mathtt{i}\mathbf{1}, b : \sharp\mathbf{1}, c : \mathtt{i}\mathbf{1}$, and assume we are using rule T-InpFree2-Wrong. We would have $\Gamma \vdash P$, for $P \triangleq ab \mid \overline{a}c \mid \overline{b}$. However, $P \longrightarrow_{\mathrm{ea}} c/b \mid \overline{b} \longrightarrow_{\mathrm{ea}} c/b \mid \overline{c}$, and the final derivative is not typable under Γ (as Γ only authorises inputs at c).

In πP, narrowing depends on the negative or positive occurrences of a name.

Theorem 4.2.1 (Polarised narrowing)**.** *Let T_1 and T_2 be two types such that $T_1 \leq T_2$.*

1. *If a occurs only positively in P, then $\Gamma, a : T_2 \vdash P$ implies $\Gamma, a : T_1 \vdash P$.*
2. *If a occurs only negatively in P, then $\Gamma, a : T_1 \vdash P$ implies $\Gamma, a : T_2 \vdash P$.*
3. *If a occurs both positively and negatively in P, then it is in general unsound to replace, in a typing $\Gamma \vdash P$, the type of a in Γ with a subtype or a supertype.*

Proof.

1. All premises involving a are of the form $\Gamma(a) \leq T$ for some T. Applying the transitivity rule for the subtyping relation suffices to conclude.
2. When a appears only negatively, premises are of the form $T \leq \Gamma(a)$. Indeed, the premise $\Gamma \vdash a : \Gamma(b)$ (in rule T-Arc) is equivalent to $\Gamma(a) \leq \Gamma(b)$ and the premise $\Gamma \vdash a : \mathtt{i}\Gamma(b)$ (in rule T-InpFree) is equivalent to $\exists T\ \ \Gamma(a) \leq \mathtt{i}T \wedge T \leq \Gamma(b)$, by covariance of $\mathtt{i}$.
3. $a : T, b : T, c : T \vdash a/b \mid b/c$ when for example $T = \mathtt{i}\sharp\mathbf{1}$, but replacing the type of b alone with $\sharp\sharp\mathbf{1}$ or $\mathtt{ii}\mathbf{1}$ is impossible, even if $\sharp\sharp\mathbf{1} \leq \mathtt{i}\sharp\mathbf{1}$ and $\mathtt{i}\sharp\mathbf{1} \leq \mathtt{ii}\mathbf{1}$.

□

Theorem 4.2.1 (specialised to prefixes) does not contradict Theorem 3.4.4, because in πP, reduction does not satisfy relation (3.3) (presented in Section 3.4).

Theorem 4.2.1 may be seen as a refinement of the standard narrowing result (Lemma 2.3.1) for name-passing calculi. In the π-calculus, for instance, a free name only has positive occurrences. Hence the usual statement of narrowing corresponds to Theorem 4.2.1(1). In an input $a(b).P$, the binder for b represents a negative occurrence, so that if b is free in P then b has both positive and negative occurrences, which means that the type of b may not be modified, as by Theorem 4.2.1(3). In contrast, Theorem 4.2.1(2) is vacuous in π, as a name b with only negative occurrences can only be found in an input $a(b).P$ where b is not free in P.

In general, in a name-passing calculus, if a name has only positive occurrences, then its type (be it declared in the typing environment, or in the binding occurrence of that name within the process) may be replaced with a subtype, and conversely for names with only negative occurrences. The type of names with both positive and negative occurrences may not be changed.

The type system for πP enjoys subject reduction:

Theorem 4.2.2 (Subject Reduction)**.** *If $\Gamma \vdash P$ and $P \longrightarrow_{\mathrm{ea}} P'$ then also $\Gamma \vdash P'$.*

The proof of Theorem 4.2.2 is straightforward, given Theorem 4.2.1 (1) combined with the variance properties of $\mathtt{i}$ and $\mathtt{o}$. The theorem also holds for the by-need version of reduction (Section 4.3).

Remark 4.2.3. The subject reduction theorem for $\pi\mathrm{P}$ remains valid if an arc $a\!\!/b$ is allowed to act not only on the subject of prefixes, but, more generally, on all positive occurrences of b (for instance occurrences of b as the object of an output prefix). Also, subject reduction does not require that the substitutions generated by an arc are performed sequentially: several substitutions could be performed simultaneously, like in implicit fusions.

The subject reduction theorem would however break if arcs acted also on negative occurrences of b, as the arc would then behave as a fusion (otherwise Theorem 3.4.3 would lead to a contradiction with the fact that the subtyping relation in $\pi\mathrm{P}$ is not trivial).

4.3 By-need semantics

The operational semantics given to $\pi\mathrm{P}$ in Section 4.1 allows arcs to act locally, at any time. The effect of an arc is irreversible: the application of an arc $a\!\!/b$ to a prefix at b commits that prefix to interact along a name that is greater than, or equal to, a in the preorder among names. A commitment may disable certain interactions, even block a prefix for ever. Consider for instance the process

$$(\boldsymbol{\nu} a, c)\,(bv.P \mid \overline{c}w.Q \mid a\!\!/b \mid c\!\!/b) \quad . \tag{4.1}$$

There is a competition between the two arcs; if the first wins, the process is deadlocked:

$$\longrightarrow_{\mathrm{ea}} \quad (\boldsymbol{\nu} a, c)\,(av.P \mid \overline{c}w.Q \mid a\!\!/b \mid c\!\!/b)$$

since a and c are unrelated in the preorder.

We consider here an alternative semantics, in which the action of arcs is not a commitment: arcs come about only when interaction occurs. For this reason we call the new semantics *by-need* (arcs act only when 'needed'), whereas we call *eager* the semantics of Section 4.1 (arcs act regardless of matching prefixes). In this semantics, as in the π-calculus, an interaction involves both a synchronisation and a substitution; however unlike in the π-calculus where the substitution is propagated to the whole term, here substitution only replaces the subject of the interacting prefixes.

4.3.1 Relations on names: preorder, joinability

The formalisation of the new semantics makes use of a preorder on names induced by arcs. An arc is *active* if it is unguarded, i.e., it is not underneath a prefix. The preorder induced by P is the least preorder $\prec$ that includes $b \prec a$ for each active arc $a\!\!/b$ in P, and similarly for the preorder induced by a context C.

We write $P \triangleright a \curlyvee b$ (this judgement is defined formally below) if $\{a, b\}$ has an upper bound in the preorder induced by P, that is, there is a name that is above both a and b; in this case we also say that a and b are *joinable*. For instance, we have $\boldsymbol{\nu} u(u\!\!/a \mid u\!\!/b \mid Q) \triangleright a \curlyvee b$.

Definition 4.3.1 (Preorder on names)**.** Conditions φ are defined by the following grammar:

$$\varphi \;\; ::= \;\; a \prec b \;\;\Big|\;\; a \curlyvee b \;\; .$$

Condition $a \prec b$ is read "a is below b", and condition $a \curlyvee b$ is read "a and b are joinable". In both cases, we have $\mathrm{n}(\varphi) = \{a, b\}$. We first define a judgement $\Gamma \vdash \varphi$, meaning that the set of conditions Γ implies condition φ, as follows:

$$\frac{}{\Gamma \vdash a \prec a}\ \vdash\text{-REFL} \qquad \frac{\varphi \in \Gamma}{\Gamma \vdash \varphi}\ \vdash\text{-IN} \qquad \frac{\Gamma \vdash b \curlyvee a}{\Gamma \vdash a \curlyvee b}\ \vdash\text{-MIRROR} \qquad \frac{\Gamma \vdash a \prec b \quad \Gamma \vdash b \prec c}{\Gamma \vdash a \prec c}\ \vdash\text{-TRANS}$$

$$\frac{\Gamma \vdash a \prec b \quad \Gamma \vdash c \prec b}{\Gamma \vdash a \curlyvee c}\ \vdash\text{-JOIN} \qquad \frac{\Gamma \vdash a \prec b \quad \Gamma \vdash b \curlyvee c}{\Gamma \vdash a \curlyvee c}\ \vdash\text{-EXTJOIN}$$

We then define a judgement $P \rhd \varphi$ on processes (we use notation $P \rhd \Gamma$ to mean that $P \rhd \varphi$ for all φ in Γ):

$$\frac{}{a\!\!/b \rhd b \prec a}\ \rhd\text{-ARC} \qquad \frac{P \rhd \Gamma \qquad \Gamma \vdash \varphi}{P \rhd \varphi}\ \rhd\text{-COMBINE} \qquad \frac{P \rhd \varphi}{P \mid Q \rhd \varphi}\ \rhd\text{-PAR-L} \qquad \frac{Q \rhd \varphi}{P \mid Q \rhd \varphi}\ \rhd\text{-PAR-R}$$

$$\frac{P \rhd \varphi \quad a \notin \mathrm{n}(\varphi)}{(\nu a)P \rhd \varphi}\ \rhd\text{-RES} \quad .$$

By construction, the judgement $P \triangleright a \prec b$ induces a relation $\{(a,b) \mid P \triangleright a \prec b\}$, which is sometimes written $\prec_P$, that is trivially a preorder on names. Relation $\{(a,b) \mid P \triangleright a \curlyvee b\}$ however is symmetric and reflexive, but generally not transitive.

Example 4.3.2 (Mediators)**.** A process $M_{fg} = (\boldsymbol{\nu} c)(c/f \mid c/g)$ acts like a *mediator*: it joins names f and g (we have $M_{fg} \triangleright f \curlyvee g$). Mediators remind us of equators in the π-calculus [HY95], or of fusions in the explicit fusion calculus, but lack the transitivity property (e.g., $M_{fg} \mid M_{gh} \triangleright f \curlyvee h$ does not hold).

4.3.2 By-need reduction

Definition 4.3.3 (By-need reduction)**.** *By-need* reduction, $P \longrightarrow_{\mathrm{bn}} P'$, is defined by the following rules:

$$\frac{R \triangleright a \curlyvee b}{R \mid ac.P \mid \overline{b}d.Q \longrightarrow_{\mathrm{bn}} R \mid P \mid d/c \mid Q}$$

$$\frac{P \longrightarrow_{\mathrm{bn}} P'}{P \mid R \longrightarrow_{\mathrm{bn}} P' \mid R} \qquad \frac{P \longrightarrow_{\mathrm{bn}} P'}{(\boldsymbol{\nu} a)P \longrightarrow_{\mathrm{bn}} (\boldsymbol{\nu} a)P'} \qquad \frac{P \equiv \longrightarrow_{\mathrm{bn}} \equiv P'}{P \longrightarrow_{\mathrm{bn}} P'} \ .$$

Relation $\Longrightarrow_{\mathrm{bn}}$ is the reflexive transitive closure of $\longrightarrow_{\mathrm{bn}}$.

While the eager semantics has simpler rules, the by-need semantics avoids 'too early commitments' on prefixes. For instance, the only immediate reduction of the process in (4.1) is

$$\longrightarrow_{\mathrm{bn}} (\boldsymbol{\nu} a, c)\,(P \mid w/v \mid Q \mid a/b \mid c/b)$$

where prefixes $bv.P$ and $\overline{c}w.Q$ interact because their subjects are joinable in the preorder generated by the two arcs.

Lemma 4.3.4 (Eager and by-need)**.** *$P \longrightarrow_{\mathrm{bn}} P'$ (by-need semantics) implies $P \Longrightarrow_{\mathrm{ea}} P'$ (eager semantics).*

Proof. Suppose the transition involves processes $ac.P$ and $\overline{b}d.Q$, thanks to a derivation of $R \triangleright a \curlyvee b$. The latter means that some name u is an upper bound of $\{a,b\}$, via some active arcs in R. Then, according to the eager semantics, we are able to rewrite the prefixed processes into $uc.P$ and $\overline{u}d.Q$. □

Note that because of the commitments performed by $\longrightarrow_{\mathrm{ea}}$, there is no correspondence in the other direction. We can still derive from Lemma 4.3.4 and Theorem 4.2.2 the soundness of typing with respect to by-need reduction:

Corollary 4.3.5 (Subject Reduction, by-need semantics)**.** *Theorem 4.2.2 holds for the by-need semantics: if $\Gamma \vdash P$ and $P \longrightarrow_{\mathrm{bn}} P'$ then also $\Gamma \vdash P'$.*

We now move to a comparison of the two semantics.

4.3.3 Barbed congruence

We contrast barbed congruence in πP under the two semantics we have given, eager and by-need. In order to define both versions of barbed congruence, we need to define *barbs*. This requires some care, as in πP the interaction of a process with its environment may be mediated by arcs. We follow Definition 2.1.3:

Definition 4.3.6. Given process P and name a, we write $P \downarrow_{\overline{a}}^{\mathrm{ea}}$ (resp. $P \downarrow_{\overline{a}}^{\mathrm{bn}}$) if $P \mid (\boldsymbol{\nu} x)ax.\overline{w} \longrightarrow_{\mathrm{ea}} P' \mid \overline{w}$ (resp. $P \mid (\boldsymbol{\nu} x)ax.\overline{w} \longrightarrow_{\mathrm{bn}} P' \mid \overline{w}$) for some P' and $x \neq w \notin \mathrm{fn}(P)$, and symmetrically for $P \downarrow_a^{\mathrm{ea}}$ and $P \downarrow_a^{\mathrm{bn}}$.

Hence $P \downarrow_{\overline{a}}$ if the offer by the environment of an action at a may be accepted by P. Remark that $P \downarrow_{\overline{a}}^{\mathrm{bn}}$ holds if $P \equiv (\boldsymbol{\nu} \tilde{c})(\overline{a}d.Q \mid R)$ for some process R, some names $\tilde{e}$ with $a \notin \tilde{e}$, and some prefix $\overline{b}d.Q$, where b satisfies $R \triangleright a \curlyvee b$. With the same notations, $P \downarrow_a^{\mathrm{ea}}$ if $b = a$.

Weak barbs are then defined in the standard way as in Section 2.1.4, as well as barbed congruence:

Definition 4.3.7. We write $\simeq_{\mathrm{ea}}$ and $\mathbin{\approx}_{\mathrm{ea}}$ (resp. $\simeq_{\mathrm{bn}}$ and $\mathbin{\approx}_{\mathrm{bn}}$) for the strong and weak versions of eager (resp. by-need) barbed congruence, following Definitions 2.1.4 and 2.1.7.

We provide below a basic proof technique for barbed congruence, so that we need only to quantify on evaluation contexts instead of any context. An evaluation context is a context in which the hole is not guarded by a prefix—in other words, evaluation contexts are the contexts that let reductions go through.

Proposition 4.3.8 (Context lemma)**.** *Suppose that $\mathcal{R}$ is closed under evaluation contexts and symmetric. Then:*

- *if $\mathcal{R}$ is closed under $\longrightarrow_{\mathrm{bn}}$ and preserves by-need barbs, then $\mathcal{R} \subseteq \simeq_{\mathrm{bn}}$,*
- *if $\mathcal{R}$ is closed under $\longrightarrow_{\mathrm{ea}}$ and preserves eager barbs, then $\mathcal{R} \subseteq \simeq_{\mathrm{ea}}$,*
- *if $\mathcal{R}$ is closed under $\Longrightarrow_{\mathrm{bn}}$ and preserves weak by-need barbs, then $\mathcal{R} \subseteq \mathbin{\approx}_{\mathrm{bn}}$,*
- *if $\mathcal{R}$ is closed under $\Longrightarrow_{\mathrm{ea}}$ and preserves weak eager barbs, then $\mathcal{R} \subseteq \mathbin{\approx}_{\mathrm{ea}}$.*

Proof. The proof is independent on the choice of reduction we have, given only a key property on πP contexts: a context C either is an evaluation context, or the following two statements hold for all P:

1. if $C[P] \longrightarrow_{\mathrm{bn}} P_1$ then for some C', $P_1 \equiv C'[P]$ and for all Q, $C[Q] \longrightarrow_{\mathrm{bn}} C'[Q]$,
2. if $C[P]{\downarrow}^{\mathrm{bn}}_a$ then $C[Q]{\downarrow}^{\mathrm{bn}}_a$ for all processes Q.

(We used the strong by-need case as an example.) The proof also relies on the fact that contexts can be composed.

Let $\mathcal{C}(\mathcal{R})$ be the context closure of $\mathcal{R}$, i.e. $\mathcal{C}(\mathcal{R}) \triangleq \{(C[P], C[Q]) \mid P \ \mathcal{R} \ Q\}$. Since $\mathcal{C}(\mathcal{R})$ is closed under contexts (since contexts can be composed) and is symmetric by construction, we only need to prove it is barbed-preserving and reduction closed:

- suppose $C[P] \ \mathcal{R}' \ C[Q]$ and $C[P]{\downarrow}^{\mathrm{bn}}_a$, then either C is an evaluation context, in which case $C[P] \ \mathcal{R} \ C[Q]$ and we can conclude by the fact $\mathcal{R}$ is barb-preserving, or 2 holds and then $C[Q]{\downarrow}^{\mathrm{bn}}_a$.
- suppose $C[P] \ \mathcal{R}' \ C[Q]$ and $C[P] \longrightarrow_{\mathrm{bn}} P_1$, then either C is an evaluation context, in which $C[P] \ \mathcal{R} \ C[Q]$ and we can conclude by the fact $\mathcal{R}$ is reduction-closed, or 1 holds and then $P_1 \equiv C'[P]$ and for all Q, $C[Q] \longrightarrow_{\mathrm{bn}} C'[Q]$, and then $P_1 \equiv \mathcal{R} \ C'[Q]$.

We just proved that $\mathcal{C}(\mathcal{R})$ is a reduction-closed barbed congruence up to $\equiv$, which implies that $\equiv\mathcal{R}'\equiv$ is a reduction-closed (because $\equiv$ is trivially reduction closed) barbed ($\equiv$ preserves barbs) congruence ($\equiv$ is a congruence). Hence $\mathcal{R} \subseteq \mathcal{C}(\mathcal{R}) \subseteq \equiv\mathcal{R}'\equiv \subseteq \simeq_{\mathrm{bn}}$. Note that this is an example of up-to technique, which will be explain in more details in Section 5.1. □

Proposition 4.3.9. *Proposition 4.3.8 also holds when $\mathcal{R}$ is only closed under parallel contexts instead of evaluation contexts.*

Proof. Suppose $\mathcal{R}$ is closed under parallel composition, reduction-closed, and preserves barbs. We prove $\mathcal{R}' = \{((\boldsymbol{\nu}\tilde{e})P, (\boldsymbol{\nu}\tilde{e})Q) \mid P \ \mathcal{R} \ Q\}$ is a reduction-closed barbed evaluation congruence up to $\equiv$, and use Proposition 4.3.8 to conclude. The closure under evaluation context is trivial (up to $\equiv$) as an evaluation context can be decomposed (up to $\equiv$) into a context of the form $(\boldsymbol{\nu}\tilde{c})(R \mid [\cdot])$. The reduction closure and the preserving of the barbs follows from the fact that they cannot be influenced by restrictions:

1. if $P \longrightarrow_{\mathrm{bn}} P'$ then $(\boldsymbol{\nu}b)P \longrightarrow_{\mathrm{bn}} (\boldsymbol{\nu}b)P'$ and if $(\boldsymbol{\nu}b)P \longrightarrow_{\mathrm{bn}} P_1$ then $P \longrightarrow_{\mathrm{bn}} P'$ and $P_1 \equiv (\boldsymbol{\nu}b)P'$,
2. $(\boldsymbol{\nu}b)P \downarrow^{\mathrm{bn}}_a$ iff $P \downarrow^{\mathrm{bn}}_b$ and $a \neq b$.

(Contrarily to the implicit fusion calculus where restrictions can trigger reductions.) □

Example 4.3.10 (Contrasting eager and by-need semantics)**.** The eager and by-need semantics of πP yield incomparable equivalences. The following two laws are valid in the by-need case, and fail in the eager case:

$$(\boldsymbol{\nu} a)a\!/\!c = 0 \qquad\qquad a \mid a = a.a \quad .$$

To see the failure of the first law in the eager semantics, consider a context $C \triangleq [\cdot] \mid (\boldsymbol{\nu} b)(b\!/\!c) \mid c \mid \overline{c}.\overline{w}$; then $C[(\boldsymbol{\nu} a)(a\!/\!c)]$ can lose the possibility of emitting at w, by reducing in two steps to $(\boldsymbol{\nu} a)(a\!/\!c \mid a) \mid (\boldsymbol{\nu} b)(b\!/\!c \mid \overline{b}.\overline{w})$, because of a commitment determined by arcs; this cannot happen for $C[0]$. In the by-need semantics, there are no early commitments, and the two processes are hence equal.

Similarly, in the eager semantics, it is possible to put $a \mid a$ in a context where two arcs rewrite each a prefix differently, while one can only rewrite the topmost prefix in $a.a$. This scenario cannot be played in the by-need semantics.

On the other hand, the following law is valid for strong (and weak) eager equivalence, but fails to hold in the by-need case:

$$(\boldsymbol{\nu} abu)(a\!/\!u \mid b\!/\!u \mid \overline{u} \mid a.\overline{w}) = (\boldsymbol{\nu} v)(\overline{v} \mid v.\tau.\overline{w} \mid v.0) \ .$$

($\tau.\overline{w}$ stands for $\boldsymbol{\nu} c(c \mid \overline{c}.\overline{w})$ with $c \neq w$). The intuition is that concurrent arcs are used on the left-hand side to implement internal choice. As a consequence of the law $(\boldsymbol{\nu} a)a\!/\!c = 0$, in the by-need case, process $b\!/\!u$ can be disregarded on the left, so that the process on the left necessarily does the output on w.

Example 4.3.10 shows that the two semantics are incomparable. Lemma 4.3.4 suggests that the eager reduction is a decomposition of the by-need reduction. We can moreover remark that the addition of dynamic operators like guarded choice is rather natural under the by-need semantics. It is delicate in the eager semantics; for instance it is unclear whether, in a process like $c\!/\!a \mid (a.P + b.Q)$, the arc should be allowed to trigger the choice.

We have introduced πP with the eager semantics for reasons of simplicity, but we find the by-need semantics more compelling. In the following, unless otherwise stated, we work under by-need, though we also indicate what we know under eager.

4.3.4 Behavioural laws

We now present some relevant examples of equalities and inequalities in the two semantics. The laws for the by-need semantics hold for strong barbed congruence, but some corresponding laws (e.g. Lemma 4.3.14) only hold in the weak case of the eager semantics—indeed, sometimes by-need reductions are implemented using several eager reductions, in light of Lemma 4.3.4.

In the case of the by-need semantics ($\simeq_{\mathrm{bn}}$), the laws are established in Lemma 4.6.1 through a notion of bisimulation that characterises $\simeq_{\mathrm{bn}}$ (Theorem 4.5.38).

In the case of the eager semantics ($\simeq_{\mathrm{ea}}$ or $\approxeq_{\mathrm{ea}}$) we provide the proofs of the laws in this section, since we do not have such a characterisation[1] for $\simeq_{\mathrm{ea}}$. Note that proofs in the eager case differ significantly from the proofs in the by-need case.

We start by establishing laws showing the input-output asymmetry of πP and the difference between πP and fusion calculi, which have this symmetry property.

Lemma 4.3.11 (Contrasting πP and fusions)**.** *We have, for by-need, eager, and explicit fusions:*

$$\begin{array}{lll}
(\boldsymbol{\nu} b, c)\overline{a}b.\overline{a}c \not\approxeq_{\mathrm{bn}} (\boldsymbol{\nu} b)\overline{a}b.\overline{a}b & (\boldsymbol{\nu} b, c)\overline{a}b.\overline{a}c \not\approxeq_{\mathrm{ea}} (\boldsymbol{\nu} b)\overline{a}b.\overline{a}b & (\boldsymbol{\nu} b, c)\overline{a}b.\overline{a}c \not\approxeq_{\mathrm{ef}} (\boldsymbol{\nu} b)\overline{a}b.\overline{a}b \\
(\boldsymbol{\nu} b, c)ab.ac \simeq_{\mathrm{bn}} (\boldsymbol{\nu} b)ab.ab & (\boldsymbol{\nu} b, c)ab.ac \simeq_{\mathrm{ea}} (\boldsymbol{\nu} b)ab.ab & (\boldsymbol{\nu} b, c)ab.ac \not\approxeq_{\mathrm{ef}} (\boldsymbol{\nu} b)ab.ab \ .
\end{array}$$

Proof, eager case. Using Proposition 4.3.9 we prove that the following relation, closed under evaluation contexts,

$$\begin{array}{rl}
\mathcal{R} \triangleq & \{(E[(\boldsymbol{\nu} b, c)(ab.ac) \quad], E[(\boldsymbol{\nu} b)(ab.ab) \quad]) \text{ for all } E\} \\
& \cup \{(E[(\boldsymbol{\nu} b, c)(d\!/\!b \mid ac)], E[(\boldsymbol{\nu} b)(d\!/\!b \mid ab)]) \text{ for all } E\} \\
& \cup \{(E[(\boldsymbol{\nu} b, c)(d\!/\!b \mid e\!/\!c)], E[(\boldsymbol{\nu} b)(d\!/\!b \mid e\!/\!b)]) \text{ for all } E\} \ \cup \ \simeq_{\mathrm{ea}}
\end{array}$$

[1] we can build a LTS such that $\xrightarrow{\tau} = \longrightarrow_{\mathrm{ea}}$ together with a bisimulation $\sim$ that is sound ($\sim \subseteq \simeq_{\mathrm{ea}}$) but it is not obvious to reach completeness ($\simeq_{\mathrm{ea}} \subseteq \sim$).

is reduction-closed as well as $\mathcal{R}^{-1}$ (it is trivially barb-preserving in the eager case, since they have the same prefix subjects). The reductions from E alone are not interesting, and each interaction between E and its content goes from one line to the next, the last one being included in $\simeq_{\mathrm{ea}}$. The important observation is that $d\not b$ and $e\not c$ can never interact with anything, because there are no b or c in subject position.

To prove $(\boldsymbol{\nu} b, c)\overline{a}b.\overline{a}c \not\approx_{\mathrm{ea}} (\boldsymbol{\nu} b)\overline{a}b.\overline{a}b$ we use the tester process $T \triangleq (\boldsymbol{\nu} d, e)(ad.ae.(\overline{e} \mid d.\overline{w}))$, yielding:

$$(\boldsymbol{\nu} b, c)\overline{a}b.\overline{a}c \mid T \longrightarrow^2_{\mathrm{ea}} (\boldsymbol{\nu} b, c, d, e)(b\not d \mid c\not e \mid \overline{e} \mid d.\overline{w}) \approx_{\mathrm{ea}} 0$$
$$(\boldsymbol{\nu} b)\overline{a}b.\overline{a}b \mid T \longrightarrow^2_{\mathrm{ea}} (\boldsymbol{\nu} b, d, e)(b\not d \mid b\not e \mid \overline{e} \mid d.\overline{w}) \Downarrow^{\mathrm{ea}}_{\overline{w}}$$

Indeed $(\boldsymbol{\nu} b)\overline{a}b.\overline{a}b$ generates $(\boldsymbol{\nu} b)(b\not d \mid b\not e)$, authorising further reductions between d and e since b can be a unifier, whereas $(\boldsymbol{\nu} b, c)(b\not d \mid c\not e)$ will never rewrite d and e into the same name. □

These laws show a difference between input and output in behavioural equalities of πP. The reason for the inequality is that the right-hand side will eventually yield $(\boldsymbol{\nu} b)(b\not d \mid b\not e)$, authorising further reductions between d and e. The left-hand side will yield $(\boldsymbol{\nu} b, c)(b\not d \mid c\not e)$ which does not allow such reductions.

For the fusion calculi, e.g. the explicit fusion calculus, the resulting processes would be $(\boldsymbol{\nu} b)(b{=}d \mid b{=}e)$ and $(\boldsymbol{\nu} b, c)(b{=}d \mid c{=}e)$ respectively bisimilar to $d{=}e$ and 0, which are not in $\approx_{\mathrm{ef}}$. The dual of these processes are not bisimilar either, by symmetry of $\approx_{\mathrm{ef}}$.

But in πP the dual processes would yield $(\boldsymbol{\nu} b)(d\not b \mid e\not b)$ and $(\boldsymbol{\nu} b, c)(d\not b \mid e\not c)$ which are both bisimilar to 0, indeed πP is not symmetric.

We now move to some behavioural properties of arcs in πP. The first law describes how to compose two arcs into one, the second states the idempotence of parallel composition on arcs.

Lemma 4.3.12 ($\simeq_{\mathrm{bn}}$ laws about arcs)**.** *When $a \neq b$ and $b \neq c$, we have:*

$$\boldsymbol{\nu} b(a\not b \mid b\not c) \simeq_{\mathrm{bn}} a\not c \qquad \boldsymbol{\nu} b(a\not b \mid b\not c) \approx_{\mathrm{ea}} a\not c$$
$$a\not b \mid a\not b \simeq_{\mathrm{bn}} a\not b \qquad a\not b \mid a\not b \simeq_{\mathrm{ea}} a\not b \ .$$

Proof, eager case. The relation induced by $a\not b \mid a\not b \simeq_{\mathrm{ea}} a\not b$ and closed under evaluation contexts is trivially reduction-closed and trivially preserves barbs. Consider now the relation induced by the first law, again closed under evaluation contexts, and close it with all the rewritings that $b\not c$ and $a\not c$ can operate on prefixes of subject c on both sides: some prefixes of the form $\overline{b}d.Q$ and $bd.Q$ on the left-hand side will correspond to $\overline{a}d.Q$ and $ad.Q$ on the right-hand side. More precisely, we write, when S is a set of prefixed processes with subject c:

$$\mathrm{LHS}_S \triangleq (\boldsymbol{\nu}\tilde{e})\Big(\ \boldsymbol{\nu} b\Big(a\not b \mid b\not c \mid \prod_{\overline{c}d.Q \in S} \overline{b}d.Q \mid \prod_{cd.Q \in S} bd.Q\Big) \mid R\Big)$$
$$\mathrm{RHS}_S \triangleq (\boldsymbol{\nu}\tilde{e})\Big(\quad a\not c \quad \mid \prod_{\overline{c}d.Q \in S} \overline{a}d.Q \mid \prod_{cd.Q \in S} ad.Q \ \mid R\Big)$$

then by construction, the relation $\{(\mathrm{LHS}_S, \mathrm{RHS}_S) \mid S\}$ is reduction-closed in the two directions:

- left to right, each rewriting from $b\not c$ can be caught up with a rewriting from $a\not c$, adding a process to the set S. Each rewriting from $a\not b$ removes a process from S, the corresponding process going into R.
- Right to left, each rewriting from $a\not c$ can be caught up with two rewritings from $b\not c$ and then from $a\not b$. The barbs are visibly the same on both sides, except some barbs on a in RHS_S that are only weak barbs from LHS_S.

□

As a consequence, any input and output of πP can be transformed into a bound prefix, by introducing a new restricted name:

Lemma 4.3.13 (From free to bound prefixes)**.** *We have, for $x \notin \mathrm{fn}(P) \cup \{a, b\}$ and $y \notin \mathrm{fn}(Q) \cup \{a, c\}$:*

$$\begin{array}{ll} ab.P \simeq_{\mathrm{bn}} (\boldsymbol{\nu} x)ax.(x/b \mid P) & ab.P \approxeq_{\mathrm{ea}} (\boldsymbol{\nu} x)ax.(x/b \mid P) \\ \overline{a}c.Q \simeq_{\mathrm{bn}} (\boldsymbol{\nu} y)\overline{a}y.(c/y \mid Q) & \overline{a}c.Q \approxeq_{\mathrm{ea}} (\boldsymbol{\nu} y)\overline{a}y.(c/y \mid Q) \ . \end{array}$$

Proof, eager case. We prove the first law for $\approxeq_{\mathrm{ea}}$. By Proposition 4.3.9 we prove $\mathcal{R} \triangleq \{(R \mid ab.P, R \mid (\boldsymbol{\nu} x)ax.(x/b \mid P))\} \cup \approxeq_{\mathrm{ea}}$ is weakly reduction-closed and preserves weak barbs. Reductions or barbs from R alone are trivially handled, what is interesting is when we get to an interaction between an output in R and the input on a, in which case $R = E[\overline{a}c.Q]$ and the resulting pair would be, up to $\equiv$, the following processes:

$$E[Q \mid c/b \mid P] \quad \approxeq_{\mathrm{ea}} \quad E[Q \mid (\boldsymbol{\nu} x)(c/x \mid x/b) \mid P] \ .$$

These are related through $\approxeq_{\mathrm{ea}}$, by Lemma 4.3.12. The proof for the output is similar. □

If these laws are applied to all inputs and outputs of a process P, then the result is a behaviourally equivalent process P', in which all names exchanged in an interaction are fresh. Thus P' reminds us of the πI variant of π (Section 3.1).

The following laws involve arcs in relation with negative or positive occurrences of names. These are reminiscent of standard results about substitutions, forwarders and equators in the asynchronous π-calculus (see, e.g., [SW01]). Note that the first law generalises the composition of arcs in Lemma 4.3.12.

Lemma 4.3.14 (Substitution and polarities)**.** *Suppose $a \neq b$.*

1. *If a has only positive occurrences in P, then $(\boldsymbol{\nu} a)(P \mid b/a) \simeq_{\mathrm{bn}} P\{b/a\}$;*
2. *if a has only negative occurrences in P, then $(\boldsymbol{\nu} a)(P \mid a/b) \simeq_{\mathrm{bn}} P\{b/a\}$;*
3. *$(\boldsymbol{\nu} a)(P \mid b/a \mid a/b) \simeq_{\mathrm{bn}} P\{b/a\}$.*

In addition, 1 and 3 hold for $\approxeq_{\mathrm{ea}}$, but 2 does not.

Proof, eager case. Item 2 does not hold for $\approxeq_{\mathrm{ea}}$ because a/b may transform positive occurrences of b into positive occurrences of a, violating the condition about a. And indeed, with $P = 0$ we already know that $(\boldsymbol{\nu} a)a/b \not\approxeq_{\mathrm{ea}} 0$ (as in the proof of Lemma 4.3.11).

We now prove items 1 and 3. We write σ for the substitution that replaces a with b. We prove that the induced relations

$$\begin{array}{l} \mathcal{R}_1 \triangleq \{((\boldsymbol{\nu} a)(b/a \mid P), P_\sigma) \mid \text{ for all } P \text{ such that } a \notin \mathrm{fn}^-(P)\} \\ \mathcal{R}_2 \triangleq \{((\boldsymbol{\nu} a)(a/b \mid b/a \mid P), P_\sigma) \mid \text{ for all } P\} \end{array}$$

are evaluation-context-closed up to structural congruence, weakly reduction-closed and preserve weak barbs. To prove closure under evaluation contexts, observe that $E[(\boldsymbol{\nu} a)(b/a \mid P)] \equiv (\boldsymbol{\nu} a)(b/a \mid P')$ for $P' = E[P]$ and $E[P_\sigma] = (E[P])_\sigma$. We consider now closure under reduction.

We write $P \xrightarrow{b/a} P'$ whenever a prefix $\overline{a}d.Q$ in P is rewritten into the corresponding $\overline{b}d.Q$ in P' and we write $\xrightarrow{a=b}$ for the symmetric closure of $\xrightarrow{b/a}$. We remark the following correspondences:

1. if $P \longrightarrow_{\mathrm{ea}} P'$ then $P_\sigma \longrightarrow_{\mathrm{ea}} P'_\sigma$;
2. if $P \xrightarrow{a=b} P'$ then $P_\sigma = P'_\sigma$;
3. if $P_\sigma \longrightarrow_{\mathrm{ea}} P_1$ then $P_1 = P'_\sigma$ for some P' such that one of the following holds:
   (a) $P \longrightarrow_{\mathrm{ea}} P'$;
   (b) $P \xrightarrow{b/a} \longrightarrow_{\mathrm{ea}} P'$, which implies $b/a \mid P \longrightarrow^*_{\mathrm{ea}} b/a \mid P'$;

(c) $P \xrightarrow{a=b} \longrightarrow_{\mathrm{ea}} P'$ (only if $a \in \mathrm{fn}^-(P)$), which implies $a/b \mid b/a \mid P \longrightarrow_{\mathrm{ea}}^* a/b \mid b/a \mid P'$.

1 and 2 are trivial. To prove 3, remark that when $P_\sigma \longrightarrow_{\mathrm{ea}} P'_\sigma$, we are in one of the following cases:

- $(\overline{c}d.Q \mid ef.R)_\sigma \longrightarrow_{\mathrm{ea}} (c/f \mid Q \mid R)_\sigma$ if $\sigma(c) = \sigma(e)$ (then, if $c \neq e$, one step of $\xrightarrow{b/a}$ can catch up);
- $(c/d \mid \overline{e}f.R)_\sigma \longrightarrow_{\mathrm{ea}} (c/d \mid \overline{c}f.R)_\sigma$ if $\sigma(d) = \sigma(e)$. Then, if $c \neq e$ then if $e = a$ and $d = b$ then one step of $\xrightarrow{b/a}$ can catch up, otherwise we first need to transform $bf_\sigma.R_\sigma$ back into $af_\sigma.R_\sigma$ with a step of $\xrightarrow{a=b}$.
- $(c/d \mid ef.R)_\sigma \longrightarrow_{\mathrm{ea}} (c/d \mid cf.R)_\sigma$ if $\sigma(d) = \sigma(e)$, handled as above.

With those correspondences, it is easy to show that $\mathcal{R}_1$ and $\mathcal{R}_2$ are reduction-closed, as well as $\mathcal{R}_1^{-1}$ and $\mathcal{R}_2^{-1}$: a reduction from the left-hand side is caught up with zero or one reduction on the right-hand side (thanks to 1 and 2), and a reduction from the right-hand side is caught up with one or two reductions from the left-hand side (thanks to 3).

Preservation of barbs follows the argument from closure under reductions. □

4.4 Private names

Lemma 4.3.11 gives an example of a law that holds in πP and not in fusion calculi. Unfortunately this law contains free input prefixes, which makes us question its significance as there is no counterpart outside single-binder calculi. We study in this section an equation that can be formulated in all name-passing calculi considered so far, as long as they feature a guarded choice operator.

We recall the *interleaving law* ((3.2) from Section 3.2):

$$\overline{a}(x).\overline{b}(y).(\overline{x} \mid y) \;=\; \overline{a}(x).\overline{b}(y).(\overline{x}.y + y.\overline{x}) \;. \tag{4.2}$$

Prefix $\overline{a}(x)$ is the emission of a fresh name x on a, i.e. it is short for $(\boldsymbol{\nu} x)\overline{a}x$. Prefix $\overline{x}$ (resp. y) stands for an output (resp. input) where the value being transmitted is irrelevant. Equation (4.2) holds for strong barbed congruence both in πP and in π, but does not hold in fusion calculi. Indeed in fusions, unlike in the π-calculus, the process that creates successively two fresh names x and y cannot prevent the context from equating ("fusing") x and y.

Hence, in order for the equivalence to hold in fusion calculi, it is necessary to add a third summand on the right, $[x{=}y]\tau$, that is triggered only when the environment entails that x and y are related in the equivalence relation:

$$\overline{a}(x).\overline{b}(y).(\overline{x} \mid y) \;\simeq_{\mathrm{ef}}\; \overline{a}(x).\overline{b}(y).(\overline{x}.y + y.\overline{x} + [x{=}y]\tau) \;.$$

(The meaning of the prefix $[x{=}y]\tau.P$ is made precise in Section 4.6.3.) This example suggests that πP provides a better control on restricted names than existing fusion calculi, in which it is hard to statically prevent the aliasing of two names.

4.4.1 Related works, privacy and symmetry

Control on privacy This issue of name privacy also partially motivated the study of two variants of the implicit fusion calculus, that feature refined notions of restriction. The distinctive fusion calculus (D) introduced by Buscemi et al. [BBM04] defines two binding operators: one corresponds to restriction in fusion calculi (authorising aliasing with other names), and the other corresponds to restriction in π (forbidding all aliasings). The U-Calculus (U), by the same authors [BBM05] defines a more refined restriction operator, with a parameter specifying the set of names with which aliasing is forbidden. Note that it is not clear how these calculi could be transported to the explicit variant of the fusion calculus.

We hope that πP provides some control on restricted names more naturally, since we did not resort to refining the restriction operator for this purpose alone.

Minimality The other difference with respect to the implicit fusion calculus is that prefixes are not polarised in Buscemi et al's calculi: they feature neither outputs nor inputs, but only one form of polyadic prefix. They recover polarities by encoding inputs and outputs using different binders on dummy variables, in such a way that any synchronisation between two inputs or two outputs would imply the fusing of two restricted names.

This calls for another informal observation about minimality and symmetry, as an addendum to Section 1.3. By definition, symmetry suggests redundancy: if the input and output constructs play the same role, couldn't we remove one of them? This indeed is the case for the previous two calculi. Another example is the Concurrent Pattern Calculus [GWGJ10] (CPC) where polarities can be recovered through a pattern unification process.

However, this observation does not hold for all symmetric calculi: some still need polarities, especially the simpler ones (like CCS, fusion-calculi, and πI). In fact, calculi for which polarities can be recovered by other constructs are arguably more intricate than the ones for which they cannot. For instance, in all three of the aforementioned calculi (D, U and CPC), deciding whether a synchronisation can happen or not depends on the occurrences of binders on both sides of the communication.

For example, in D and U, process P_1 below can have a reduction iff x' and y' are indeed x and y: if there is no λ-binding of either one of x and y, the reduction cannot happen. In CPC, process P_2 below has a reduction only if the pattern compounds p_1 and p_2 contain no binders.

$$P_1 = (\lambda x')(\nu y)a\langle x, y\rangle.P \mid (\lambda y')(\nu x)a\langle x, y\rangle.Q \qquad\qquad P_2 = a\langle \lambda x \bullet p_1\rangle.P \mid a\langle p_2 \bullet \lambda y\rangle.Q$$

4.5 Coinductive characterisation of barbed congruence

The behavioural theory of πP does not seem to be reducible to existing calculi: novel properties are the fact that the preorder relation induces a communicability relation[2] that is not transitive, and that positive private names are staying private, despite the free inputs. In particular, name preorders have an impact on how processes express behaviours. Some laws for barbed congruence in πP show that the behavioural theory of πP departs from existing fusion calculi (Lemma 4.3.11 and Section 4.4).

The main purpose of this section is to deepen the study of the behavioural theory of πP, in an untyped setting. We define a Labelled Transition System for πP, and show that the induced notion of bisimilarity. The LTS reveals interesting aspects of interaction in πP. An important observation is related to the interplay between arcs and the restriction operator. It is for instance possible for a process to react to an input offer on some channel, say c, without actually being able to perform an output on c. This is the case for the following process:

$$P_0 \triangleq (\boldsymbol{\nu} a)(\overline{a}v \mid a\!\not\!c) \qquad\qquad P_0 \mid cu \longrightarrow v\!\not\!u \ . \tag{4.3}$$

(note that P_0 could do an output on c if the arc $a\!\not\!c$ was replaced with $c\!\not\!a$). This phenomenon leads to the addition of a new type of labels in the LTS, corresponding to what we call *protected actions* (accordingly, we introduce *protected names*, which correspond to (usages of) names where a protected action occurs: intuitively, name c is protected in P_0). As expected, protected actions correspond to observables of barbed congruence.

In addition, arc processes do not have transitions, but they induce relations between names, which in turn influence the behaviour of processes. Accordingly, strong bisimilarity not only tests transitions, but also has a clause to guarantee that related processes entail the same conditions.

Finally, the LTS also includes a label $[\varphi]\tau$, expressing "conditional synchronisation". Intuitively, process $\overline{a} \mid b$ is not able to perform a τ transition by itself, but it should be when the environment entails $a \curlyvee b$. Hence, in order for our LTS to be compositional, we include labels of the form $[\varphi]\tau$, interpreted as "τ under the condition φ".

[2] given a process, this is the relation on names that relate a to b when an input on a can communicate with an output on b. In πP, it is $\curlyvee_P$. In π and implicit fusions, it is the identity relation.

We give in Section 4.6 an axiomatisation of barbed congruence. For this, we introduce the guarded sum construct and a prefix corresponding to label $[\varphi]\tau$ akin to the matching construct in π—this is customary for axiomatisations for the π-calculus [PS95, SW01]. The resulting calculus extended with sum and this prefix is called $\pi\mathrm{P}^+$. These two new constructs cannot be encoded in a simple way in the original calculus $\pi\mathrm{P}$. Fortunately they add no discriminative power: the induced behavioural equivalences coincide (Proposition 4.5.39). In light of this observation, this section builds an LTS for $\pi\mathrm{P}^+$, as it is the more general calculus.

To simplify the presentation of the axiomatisation, we renounce minimality even more, using two kinds of bound prefixes (bound prefixes are also customary for axiomatisations[3]). This particular addition is unimportant, as there are trivial encodings in the two directions (Remark 4.5.4). We discuss in Section 4.5.6 a presentation of transitions and bisimilarity based on free prefixes.

As usual, we focus on a finite calculus. This is sufficient to enlighten the main aspects of the behavioural theory of processes. We do not expect any unpredicted difficulty to arise, in the definition of labelled transitions and bisimilarity, from the extension of $\pi\mathrm{P}^+$ with a replication operator.

4.5.1 Extending the calculus

We describe now the additional constructs of $\pi\mathrm{P}^+$: name are replaced with *extended names* (α, β), prefixes (π) now feature a third construct, and processes now feature sums:

$$\alpha \;::=\; a \;\mid\; \{a\} \qquad \pi \;::=\; \alpha(x) \;\mid\; \overline{\alpha}(x) \;\mid\; [\varphi]\tau \qquad P \;::=\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \sum_{i\in I} \pi_i.P_i \;\mid\; a\!\!/b \;.$$

The new constructs are sums, conditional τs, extended names, and bound prefixes (as explained above, we see in Remark 4.5.4 that the latter two do not change expressiveness, and in Proposition 4.5.39 that barbed congruence is preserved with all the new constructs). Relations $\prec$ and $\curlyvee$ extend to extended names, which we define and explain in the following, where needed. When $\mathrm{n}(\varphi) = \{a\}$, we say that φ is *reflexive* and we abbreviate prefix $[\varphi]\tau$ as τ.

In a prefix $\alpha(x)$ or $\overline{\alpha}(x)$, we say that extended name α is in subject position, while x is in object position. As discussed previously, extended names include *protected names*, of the form $\{a\}$, which can be used in subject position only. We call *protected prefix* a prefix where the subject is a protected name. A prefix of the form $[\varphi]\tau$ is called a *conditional* τ and other prefixes are called *visible*.

Bound and free names for prefixes are given by:

$$\mathrm{bn}([\varphi]\tau) \triangleq \emptyset \qquad \mathrm{bn}(\alpha(x)) \triangleq \mathrm{bn}(\overline{\alpha}(x)) \triangleq \{x\}$$

$$\mathrm{fn}([\varphi]\tau) \triangleq \mathrm{n}(\varphi) \qquad \mathrm{fn}(\alpha(x)) \triangleq \mathrm{fn}(\overline{\alpha}(x)) \triangleq \mathrm{n}(\alpha) \text{ with } \mathrm{n}(a) \triangleq \mathrm{n}(\{a\}) \triangleq \{a\} \;.$$

In a sum process, we let I range over a finite set of integers. 0 is the inactive process, defined as the empty sum. We use S to range over sum processes of the form $\sum_{i\in I} \pi_i.P_i$, and write $\pi.P \in S$ if $\pi.P$ is a summand of S. We sometimes decompose sum processes using the binary sum operator, writing, e.g., $S_1 + S_2$ (in particular, $S + 0 = S$). As previously, we abbreviate $\pi.0$ as π, and write $\alpha(x).P$ simply as $\alpha.P$ when the transmitted name is not relevant, and similarly for $\overline{\alpha}$. In $(\boldsymbol{\nu} a)P$, $(\boldsymbol{\nu} a)$ binds a in P, and prefixes $\alpha(x)$ and $\overline{\alpha}(x)$ bind x in the continuation process. As usual, binders give rise to α-conversion.

We use an overloaded notation, and define processes $a \curlyvee b$ and $a \prec b$ as follows, to represent conditions:

$$a \curlyvee b \triangleq (\boldsymbol{\nu} u)(u\!\!/a \mid u\!\!/b) \qquad a \prec b \triangleq b\!\!/a \;.$$

Since Definition 4.3.1 does not mention prefixes at all, there is no need to adapt the definitions of $\Gamma \vdash \varphi$ and $P \rhd \varphi$ from $\pi\mathrm{P}$ to $\pi\mathrm{P}^+$. However, we will use the following definition: $\Phi(P) \triangleq \{\varphi \mid P \rhd \varphi\}$ is the set of all conditions entailed by P.

[3] It can be noted that the axiomatisation of fusions given in [PV98a] relies only on free input and output, and treats bound prefixes as derived operators. We think that, for $\pi\mathrm{P}$, handling prefixes for bound and protected actions as derived operators would introduce further technical complications that would make the axiomatisation more obscure.

Positive and negative occurrences of names can be extended to extended names and conditional synchronisation prefixes:

Definition 4.5.1 (Positive and negative occurrences)**.** In protected prefixes, subjects are negative occurrences. In conditions, names are negative except in $a \prec b$ where b is positive. In $[\varphi]\tau$ the polarities are opposite to the one in φ. More formally, we extend $\mathrm{fn}^+(P)$ and $\mathrm{fn}^-(P)$ as follows:

$$\mathrm{fn}^+(a) \triangleq \{a\} \qquad \mathrm{fn}^+(\{a\}) \triangleq \emptyset \qquad \mathrm{fn}^-(a) \triangleq \emptyset \qquad \mathrm{fn}^-(\{a\}) \triangleq \{a\}$$

$$\mathrm{fn}^+([\varphi]\tau.P) \triangleq \mathrm{fn}^-(\varphi) \cup \mathrm{fn}^+(P) \qquad \mathrm{fn}^+(a \prec b) \triangleq \{b\} \qquad \mathrm{fn}^+(a \curlyvee b) \triangleq \emptyset$$

$$\mathrm{fn}^-([\varphi]\tau.P) \triangleq \mathrm{fn}^+(\varphi) \cup \mathrm{fn}^-(P) \qquad \mathrm{fn}^-(a \prec b) \triangleq \{a\} \qquad \mathrm{fn}^-(a \curlyvee b) \triangleq \{a, b\}$$

$$\mathrm{fn}^+(\alpha(x).P) \triangleq \mathrm{fn}^+(\overline{\alpha}(x).P) \triangleq \mathrm{fn}^+(\alpha) \cup (\mathrm{fn}^+(P) \setminus \{x\})$$

$$\mathrm{fn}^-(\alpha(x).P) \triangleq \mathrm{fn}^-(\overline{\alpha}(x).P) \triangleq \mathrm{fn}^-(\alpha) \cup (\mathrm{fn}^-(P) \setminus \{x\})$$

Definition 4.5.2. Structural congruence, written $\equiv$, is the smallest congruence satisfying the laws of Definition 2.1.1 and the following law handling the sum construct:

$$\sum_{i \in I} \pi_i.P_i \equiv \sum_{i \in I} \pi_{\sigma(i)}.P_{\sigma(i)} \quad \text{when } \sigma \text{ a permutation of } I \ .$$

Relations $\equiv$ and $\rhd$ are used to define the reduction of processes. We rely on $\rhd$ to infer that two processes interact on joinable (extended) names. This allows us to introduce the reduction relation $\longrightarrow$ (adapted from $\longrightarrow_{\mathrm{bn}}$ in Definition 4.3.3 to handle choice and conditional τs):

Definition 4.5.3 (Reduction)**.** Relation $\longrightarrow$ is the relation defined by the following rules:

$$\frac{\overline{\alpha}(x).P \in S_1 \quad \beta(y).Q \in S_2 \quad R \rhd \alpha \curlyvee \beta \quad x \neq y}{R \mid S_1 \mid S_2 \ \longrightarrow \ R \mid (\boldsymbol{\nu} xy)(x/y \mid P \mid Q)} \qquad \begin{array}{l} a \curlyvee \{b\} = \{b\} \curlyvee a = a \prec b \\ \{a\} \curlyvee \{b\} = \text{undefined} \end{array}$$

$$\frac{[\varphi]\tau.P \in S \qquad R \rhd \varphi}{R \mid S \ \longrightarrow \ R \mid P} \qquad \frac{P \longrightarrow P'}{P \mid R \longrightarrow P' \mid R} \qquad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} a)P \longrightarrow (\boldsymbol{\nu} a)P'} \qquad \frac{P \equiv \longrightarrow \equiv P'}{P \longrightarrow P'}$$

Again, we use the generic definition of barbs, uniform over usual process calculi, and define $P \downarrow_{\overline{a}}$ as $P \downarrow_{\overline{a}}^{\mathrm{bn}}$ in Definition 4.3.6, i.e. $P \downarrow_{\overline{a}}$ iff $P \mid a(x).\overline{w} \longrightarrow \overline{w} \mid P'$ for some P' and $w \notin \mathrm{fn}(P)$ (similarly for $P \downarrow_a$). We can remark that $P_0 \downarrow_{\overline{c}}$, where $P_0 \triangleq (\boldsymbol{\nu} a)(\overline{a}(v) \mid a/c)$ is the bound prefix version of process P_0, previously defined. Barbed congruence, now written $\simeq$, is defined as in Definition 2.1.4 over the reduction and barbs defined above (same definition as $\simeq_{\mathrm{bn}}$).

Remark 4.5.4 (Encodability of free and protected prefixes)**.** In πP, arcs act like "instantaneous forwarders". This allows us to define an encoding $[\cdot]_{\mathrm{f}}$ from a calculus with free prefixes to a calculus with bound prefixes as follows (x is chosen fresh):

$$[ab.P]_{\mathrm{f}} \triangleq a(x).([P]_{\mathrm{f}} \mid x/b) \qquad [\overline{a}b.P]_{\mathrm{f}} \triangleq \overline{a}(x).([P]_{\mathrm{f}} \mid b/x) \ ,$$

where $[\cdot]_{\mathrm{f}}$ preserves other operators of the calculi.

We return to this encoding below (Proposition 4.5.39), and show that it allows us to reflect behavioural equivalence of Definition 4.3.7 into this variant of the calculus.

We can also encode protected prefixes as follows (u is chosen fresh):

$$[\{a\}(x).P]_{\mathrm{p}} \triangleq (\boldsymbol{\nu} u)(u/a \mid u(x).[P]_{\mathrm{p}}) \qquad [\{\overline{a}\}(x).P]_{\mathrm{p}} \triangleq (\boldsymbol{\nu} u)(u/a \mid \overline{u}(x).[P]_{\mathrm{p}}) \ .$$

Although protected prefixes are in some sense redundant, we do not treat them as derived operators, to simplify the presentation (in particular in Section 4.6).

$$\frac{x \notin \mathrm{n}(\alpha) \cup \{y\} \cup \mathrm{fn}(P)}{\alpha(y).P \xrightarrow{\alpha(x)} (\boldsymbol{\nu} y)(x/y \mid P)}\ \rightarrow\text{-IN} \qquad \frac{x \notin \mathrm{n}(\alpha) \cup \{y\} \cup \mathrm{fn}(P)}{\overline{\alpha}(y).P \xrightarrow{\overline{\alpha}(x)} (\boldsymbol{\nu} y)(y/x \mid P)}\ \rightarrow\text{-OUT} \qquad \frac{}{[\varphi]\tau.P \xrightarrow{[\varphi]\tau} P}\ \rightarrow\text{-TAU}$$

$$\frac{P \xrightarrow{\overline{\alpha}(x)} P' \qquad Q \xrightarrow{\beta(x)} Q'}{P \mid Q \xrightarrow{[\alpha \curlyvee \beta]\tau} (\boldsymbol{\nu} x)(P' \mid Q')}\ \rightarrow\text{-COMM-L} \qquad \frac{P \xrightarrow{[\varphi_2]\tau} P' \quad P \rhd \Gamma \quad \Gamma, \varphi_1 \vdash \varphi_2}{P \xrightarrow{[\varphi_1]\tau} P'}\ \rightarrow\text{-TAU-}\rhd$$

$$\frac{P \xrightarrow{\alpha(x)} P' \quad P \rhd \alpha \prec \beta}{P \xrightarrow{\beta(x)} P'}\ \rightarrow\text{-IN-}\rhd \qquad \frac{P \xrightarrow{\overline{\alpha}(x)} P' \quad P \rhd \alpha \prec \beta}{P \xrightarrow{\overline{\beta}(x)} P'}\ \rightarrow\text{-OUT-}\rhd \qquad \begin{array}{l} a \prec \{b\} = a \curlyvee b \\ \{a\} \prec \{b\} = b \prec a \\ \{a\} \prec b = \text{undefined} \end{array}$$

$$\frac{P \xrightarrow{\mu} P' \qquad a \notin \mathrm{n}(\mu)}{(\boldsymbol{\nu} a)P \xrightarrow{\mu} (\boldsymbol{\nu} a)P'}\ \rightarrow\text{-RES} \qquad \frac{P \xrightarrow{\mu} P' \qquad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \rightarrow\text{-PAR-L} \qquad \frac{\pi_i.P_i \xrightarrow{\mu} P'}{\sum_i \pi_i.P_i \xrightarrow{\mu} P'}\ \rightarrow\text{-SUM}$$

Table 4.1: LTS for $\pi\mathrm{P}^{+}$. Symmetric versions of the two rules involving $\mid$ are omitted.

4.5.2 Labelled transition system

The LTS defines transitions $P \xrightarrow{\mu} P'$, where the grammar for the labels, μ, is the same as the one for prefixes π. We comment on the rules, given on Table 4.1.

The first two rules correspond to the firing of visible prefixes. The transition involves a fresh name x, upon which the participants in a communication "agree". Name y remains local, via the installation of an arc, according to the polarity of the prefix. (Adopting a rule with no arc installation would yield a more complex definition of $\sim$). The rule for the $[\varphi]\tau$ prefix is self explanatory. The rule describing communication follows the lines of the corresponding rule for $\longrightarrow$; no arc is installed (but arcs are introduced in the prefix rules).

The three rules mentioning $\rhd$ are called *preorder rules*. The two preorder rules for visible actions exploit $\prec$, which is defined for extended names (as we did for $\curlyvee$ above). Note that the condition involving $\rhd$ is *the same* in these two rules. For instance, if $P \xrightarrow{a(x)} P'$ and $P \rhd a \prec c$, we can derive $P \xrightarrow{c(x)} P'$. In this case, $P \rhd a \prec \{c\}$ is also derivable, hence $P \xrightarrow{\{c\}(x)} P'$. We can also check that P'_0, a counterpart of P_0 in (4.3), can do a transition labelled with $\{\overline{c}\}(y)$:

$$P'_0 \triangleq (\boldsymbol{\nu} a)(\overline{a}(x).0 \mid a/c) \qquad\qquad P'_0 \xrightarrow{\{\overline{c}\}(y)} (\boldsymbol{\nu} a)((\boldsymbol{\nu} x)x/y \mid a/c)$$

because $\overline{a}(x).0 \mid a/c \rhd c \curlyvee a$ (and hence $a \prec \{c\}$).

The other preorder rule can be used to modify conditional τs involved in a transition. As an example, let

$$P_1 \triangleq (\overline{a}(x).Q \mid n/u) \ \mid\ (u(y).R \mid n/a)\ .$$

Process P_1 can perform a τ transition: the two arcs can, intuitively, let the output at a and the input at u interact at name n. Technically, this can be derived by inferring a $\xrightarrow{[a \curlyvee u]\tau}$ transition (from the output on the left and the input on the right), which can then be turned into a τ transition, exploiting the fact that *the whole process* entails $a \curlyvee u$.

Finally, the congruence rules, on the last line of Table 4.1, are as expected.

Definition 4.5.5 ($\sim$)**.** A symmetric relation $\mathcal{R}$ is a bisimulation if $P \ \mathcal{R}\ Q$ implies:

- If $P \rhd \varphi$ then $Q \rhd \varphi$.
- If $P \xrightarrow{\alpha(x)} P'$, with $x \notin \mathrm{fn}(Q)$, then there is Q' such that $Q \xrightarrow{\alpha(x)} Q'$ and $P' \ \mathcal{R} \ Q'$; we impose the same condition with $\overline{\alpha}$ instead of α.
- If $P \xrightarrow{[\varphi]\tau} P'$ then there is Q' such that $Q \xrightarrow{[\varphi]\tau} Q'$ and $P' \mid \varphi \ \mathcal{R} \ Q' \mid \varphi$.

Bisimilarity, written $\sim$, is the greatest bisimulation.

In the last clause for $[\varphi]\tau$, we add φ in parallel, since the transition is fired only if φ is satisfied. Indeed demanding $P' \ \mathcal{R} \ Q'$ is too strong a requirement: arcs are permanent processes, so a context triggering such a transition necessarily entails φ (even after reductions). This definition can be related to the efficient bisimulation from [WG04].

Remark 4.5.6 (Scope extrusion)**.** Our LTS does not have rules for opening and closing the scope of a restriction. Instead, we rely on arcs in πP to handle scope extrusion. To see this, consider the following πP transition where a private name c is emitted:

$$\overline{a}(c).P \xrightarrow{\overline{a}(x)} (\boldsymbol{\nu} c)(c \not x \mid P) \ .$$

(Intuitively, process $\overline{a}(c).P$ has a restriction on c, since this name is private.) Name x is visible in the label, and arc $c \not x$ is installed. Through x, the environment can affect c, so that πP actually *implements* scope extrusion via arcs, without the need to move restrictions. We have:

$$\begin{aligned} \overline{a}(c).P \mid a(y).Q &\xrightarrow{\tau} (\boldsymbol{\nu} x)((\boldsymbol{\nu} c)(c \not x \mid P) \mid (\boldsymbol{\nu} y)(x \not y \mid Q)) \\ &\simeq (\boldsymbol{\nu} c)(\boldsymbol{\nu} y)(P \mid c \not y \mid Q) \ . \end{aligned}$$

In particular, in the encoding from the π-calculus to πP given in Section 4.7.2, when a fresh name is emitted, it is extruded, and we have ($[\![P]\!]$ is the encoding of P):

$$[\![a(y).Q \mid (\boldsymbol{\nu} b)\overline{a}b.P]\!] \xrightarrow{\tau} \sim [\![(\boldsymbol{\nu} b)(Q\{b/y\} \mid P)]\!] \ .$$

4.5.3 Towards the characterisation theorem

In order to present the proof that barbed congruence coincides with the bisimilarity induced by our LTS, we first need to establish several preliminary technical results, about relations $\vdash$, $\rhd$ and $\sqsubseteq$.

Mechanisation of some proofs. Some of the proofs involving tedious case enumerations have been verified using the Coq proof system [Coq]. When this is the case, the proof script files we mention are available from [Mad14]. Note that these proofs have little interest, as they are straightforward, even though they have a lot of cases.

On the $\vdash$ relation. To handle some case analyses, we decompose $\vdash$, using an additional relation, written $\Vdash$. The idea is that the transitive closure of $\Vdash$ is $\vdash$.

The predicate $\varphi_1, \varphi_2 \Vdash \varphi_3$ is defined as follows (note how the second line is the symmetric variant of the first):

$$\frac{}{a \prec b, b \prec c \Vdash a \prec c} \qquad \frac{}{a \prec c, b \prec c \Vdash a \curlyvee b} \qquad \frac{}{a \curlyvee b, c \prec a \Vdash c \curlyvee b} \qquad \frac{}{a \curlyvee b, c \prec b \Vdash a \curlyvee c}$$

$$\frac{}{b \prec c, a \prec b \Vdash a \prec c} \qquad \frac{}{b \prec c, a \prec c \Vdash a \curlyvee b} \qquad \frac{}{c \prec a, a \curlyvee b \Vdash c \curlyvee b} \qquad \frac{}{c \prec b, a \curlyvee b \Vdash a \curlyvee c}$$

We write $\varphi_1 \Vdash_{\varphi_2} \varphi_3$ whenever $\varphi_1, \varphi_2 \Vdash \varphi_3$ and we write $\Vdash_P$ for the union of all relations $\Vdash_\varphi$ between conditions for $P \rhd \varphi$. Intuitively, $\Vdash_P$ denotes all the implications between conditions that hold because of the conditions that are entailed by process P in one step.

Lemma 4.5.7 shows that if φ can be decomposed, then so can $\Vdash_\varphi$.

Lemma 4.5.7. *If $\varphi_1, \varphi_2 \Vdash \varphi$ then $\Vdash_\varphi \subseteq (\Vdash_{\varphi_1} \Vdash_{\varphi_2} \cup \Vdash_{\varphi_2} \Vdash_{\varphi_1})$.*

Proof. By case analysis on the $\Vdash$ predicate (see proof script in [Mad14].) □

Relations $\vdash$ and $\Vdash$ relate the same conditions:

Lemma 4.5.8. *If $\varphi_1, \varphi_2 \Vdash \varphi_3$ then $\varphi_1, \varphi_2 \vdash \varphi_3$. Conversely, if $\Gamma \vdash \varphi$ then $\psi \Vdash_{\psi_1} \ldots \Vdash_{\psi_n} \varphi$ for some reflexive ψ and for some $\psi_1, \ldots, \psi_n \in \Gamma$.*

Proof. The first part is trivial: each rule for $\Vdash$ is simulated by (three) rules for $\vdash$. The second part follows from the fact that $(\Vdash_\varphi) \subseteq (\Vdash_{\psi_1} \ldots \Vdash_{\psi_n} \Vdash_\zeta)$ for some $\psi_1, \ldots, \psi_n \in \Gamma$ and some reflexive ζ, which we prove by induction on $\Gamma \vdash \varphi$ (Lemma 4.5.7 covers the rules $\vdash$-TRANS, $\vdash$-JOIN, $\vdash$-EXTJOIN). □

Any number of iterations of relation $\Vdash_P$ can be reduced to two:

Lemma 4.5.9. $(\Vdash_P)^* \subseteq (\Vdash_P \Vdash_P)$.

Proof sketch. Suppose $\varphi_1 \Vdash_P \ldots \Vdash_P \varphi_3$. One obtains φ_3 from φ_1 by appending ψs such that $P \rhd \psi$ at the left or at the right of φ_1. We can in fact append all ψs that go to the left at first (an operation that can be done in one $\Vdash_P$ step), and once this is done, appending all ψs that go to the right (the other $\Vdash_P$ step). (See proof script in [Mad14].) □

In fact, we need to refine Lemma 4.5.9 with a better control on names:

Lemma 4.5.10. *Suppose $\varphi_1 \Vdash_P^* \varphi_4$ and $\mathrm{n}(\varphi_i) = \{a_i, b_i\}$. There are φ_2, φ_3 such that $\varphi_1 \Vdash_P \varphi_2 \Vdash_P \varphi_4$ and $\varphi_1 \Vdash_P \varphi_3 \Vdash_P \varphi_4$, with $\mathrm{n}(\varphi_2) = \{a_1, b_4\}$ and $\mathrm{n}(\varphi_3) = \{a_4, b_1\}$.*

Proof sketch. The proof follows the lines of the proof of Lemma 4.5.9. □

Relation $\Vdash_{P|Q}$ can be decomposed into $\Vdash_P$ and $\Vdash_Q$.

Lemma 4.5.11. $\Vdash_{P|Q} \subseteq (\Vdash_P \cup \Vdash_Q)^*$.

Proof. We prove by induction on $P \mid Q \rhd \varphi_2$ that for all φ_1 and φ_3, if $\varphi_1, \varphi_2 \Vdash \varphi_3$ then $\varphi_1 (\Vdash_P \cup \Vdash_Q)^* \varphi_3$. There are three cases. For $\rhd$-PAR-L we know that $P \rhd \varphi_2$ so $\varphi_1 \Vdash_P \varphi_3$ and for $\rhd$-PAR-R we get in the same way $\varphi_1 \Vdash_Q \varphi_3$. The case for the rule $\rhd$-COMBINE is handled using Lemma 4.5.8. □

On the $\rhd$ relation. Lemmas 4.5.12, 4.5.13 and 4.5.14 are proved using simple inductions on the $\rhd$ predicate.

Lemma 4.5.12. *If $P \rhd \varphi$ and $\mathrm{n}(\varphi) \setminus \mathrm{fn}(P) \neq \emptyset$, then φ is reflexive.*

Below, C ranges over contexts with one hole.

Lemma 4.5.13. *If $\Phi(P) \subseteq \Phi(Q)$ then $\Phi(C[P]) \subseteq \Phi(C[Q])$.*

Lemma 4.5.14. *If $P \rhd \varphi$ then $\Phi(P \mid \varphi) = \Phi(P)$.*

Lemma 4.5.15. *If $P \equiv Q$ then $\Phi(P) = \Phi(Q)$.*

Proof. By induction on the derivation of $P \equiv Q$. In most cases we show $\Phi(P) \subseteq \Phi(Q)$ and $\Phi(Q) \subseteq \Phi(P)$ separately, and by induction on $\rhd$ (we assume the last rule is not $\rhd$-COMBINE, since this case is immediate).

We focus on the following cases involving restrictions:

1. $(\boldsymbol{\nu} a)P \equiv (\boldsymbol{\nu} b)P\{b/a\}$ when $b \notin \mathrm{fn}(P)$: the last rule is $\rhd$-RES. To prove $(\boldsymbol{\nu} b)P\{b/a\} \rhd \varphi$ we can then apply $\rhd$-RES and we end up proving that if $P \rhd \varphi$ then $P\{b/a\} \rhd \varphi$ when $a, b \notin \mathrm{n}(\varphi)$. More generally $P \rhd \varphi \Rightarrow P_\sigma \rhd \varphi_\sigma$ for all substitutions σ (injective or not).

2. $(\boldsymbol{\nu} aP) \mid Q \equiv \boldsymbol{\nu} a(P \mid Q)$ with $a \notin \mathrm{fn}(Q)$. If the last rule is $\rhd$-PAR-R, coming from $Q \rhd \varphi$, we conclude easily. If the last rule is $\rhd$-PAR-L, coming from $\boldsymbol{\nu} aP \rhd \varphi$, we get $P \rhd \varphi$ from which $(\boldsymbol{\nu} a)(P \mid Q) \rhd \varphi$ follows. It is worth noting that if $a \in \mathrm{n}(\varphi)$, φ must be trivial ($a \curlyvee a$ or $a \prec a$).

3. $\boldsymbol{\nu} a(P \mid Q) \equiv (\boldsymbol{\nu} aP) \mid Q$ with $a \notin \mathrm{fn}(Q)$. Intuitively, we need to put together all contributions from P to the condition φ so that they are independent of Q. First, as before, from $\boldsymbol{\nu} a(P \mid Q) \rhd \varphi$ we get $P \mid Q \rhd \varphi$.

   We treat the case where $\varphi = b_0 \curlyvee c_0$ (the other cases are simpler). The derivation of $P \mid Q \rhd \varphi$ is built using several usages of $\rhd$-COMBINE involving processes P and Q, which we summarise as follows: there exist $b_0, \ldots, b_n$ and $c_0, \ldots, c_m$ such that for each $\zeta \in \{b_n \curlyvee c_m, b_0 \prec b_1, \ldots, b_{n-1} \prec b_n, c_0 \prec c_1, \ldots, c_{m-1} \prec c_m\}$, we have either $P \rhd \zeta$ or $Q \rhd \zeta$. We eliminate occurrences of a in the b_is and the c_js. For this, we proceed as follows:

   (a) when $a \in \mathrm{n}(\zeta)$ we replace $Q \rhd \zeta$ with $P \rhd \zeta$ (using Lemma 4.5.12 as $a \notin \mathrm{fn}(Q)$).

   (b) Hence, if $b_i = a$ then $P \rhd b_{i-1} \prec a$ and $P \rhd a \prec b_{i+1}$, so using $\rhd$-COMBINE we replace these two judgements with $P \rhd b_{i-1} \prec b_{i+1}$.
   If $i = n$, since $P \rhd b_n \curlyvee c_m$ and $a = b_n$, we deduce $P \rhd b_{n-1} \curlyvee c_m$ (the case $i = 0$ is not possible since $a \notin \mathrm{n}(\phi)$).

   (c) We continue as long as a appears in any of the b_i or c_j.

   We are left with a set of formulas ζ (entailing φ) such that $a \notin \mathrm{n}(\zeta)$ and either $P \rhd \zeta$ or $Q \rhd \zeta$, and hence $(\boldsymbol{\nu} a)P \mid Q \rhd \zeta$. From this we get $(\boldsymbol{\nu} a)P \mid Q \rhd \varphi$ by applying rule $\rhd$-COMBINE $n + m$ times.

□

On the $\sqsubseteq$ relation.

Definition 4.5.16. We define a relation $\sqsubseteq^{\varphi}$ between labels as follows: (i) $\alpha_1(x) \sqsubseteq^{\varphi} \alpha_2(x)$ and $\overline{\alpha}_1(x) \sqsubseteq^{\varphi} \overline{\alpha}_2(x)$ when $\varphi = \alpha_2 \prec \alpha_1$, and (ii) $[\varphi_1]\tau \sqsubseteq^{\varphi} [\varphi_2]\tau$ when $\varphi_1, \varphi \vdash \varphi_2$.We write $\sqsubseteq_P$ for the smallest preorder containing all $\sqsubseteq^{\varphi}$ when $P \rhd \varphi$.

Intuitively, $\eta \sqsubseteq_P \mu$ means that label μ is less general than η, given some condition (φ above) enforced by P. For instance, we have $\{a\}(x) \sqsubseteq_0 a(x)$. This notion is used in Lemma 4.5.25 to reason about the transition of processes.

Similarly as for $\Vdash_{\varphi}$, when φ can be decomposed, so can $\sqsubseteq^{\varphi}$:

Lemma 4.5.17. *If $\varphi_1, \varphi_2 \vdash \varphi$ then $(\sqsubseteq^{\varphi}) \subseteq (\sqsubseteq^{\varphi_1} \sqsubseteq^{\varphi_2}) \cup (\sqsubseteq^{\varphi_2} \sqsubseteq^{\varphi_1})$.*

Proof. We suppose $\mu \sqsubseteq^{\varphi} \eta$. If μ is a conditional τ, Lemma 4.5.7 is enough to conclude. Otherwise, we make an exhaustive case analysis on $\varphi_1, \varphi_2 \vdash \varphi$ and $\mu \sqsubseteq^{\varphi} \eta$, for instance: if $a \prec c, b \prec c \vdash a \curlyvee b$ and $\{a\}(x) \sqsubseteq^{a \curlyvee b} b(x)$ then $\{a\}(x) \sqsubseteq^{a \prec c} c(x)$ and $c(x) \sqsubseteq^{b \prec c} b(x)$. (The complete case analysis appears in the proof script [Mad14].) □

We show that on extended names, $\prec$ is transitive and $\curlyvee$ can be extended on the left (or on the right), but cannot in general be derived from $\prec$. From this lemma, we get tools to manipulate the $\sqsubseteq_P$ relation.

Lemma 4.5.18 (Composing relations $\prec$ and $\curlyvee$)**.**

1. *If $\alpha \prec \beta$ and $\beta \prec \gamma$ are defined, then so is $\alpha \prec \gamma$ and $\alpha \prec \beta, \beta \prec \gamma \vdash \alpha \prec \gamma$.*

2. *If $\alpha \prec \beta$ and $\beta \curlyvee \gamma$ are defined, then so is $\alpha \curlyvee \gamma$ and $\alpha \prec \beta, \beta \curlyvee \gamma \vdash \alpha \curlyvee \gamma$.*

3. *In general $\alpha \prec \beta$ and $\gamma \prec \beta$ do not imply $\alpha \curlyvee \gamma$.*

Proof. The first two items are trivial after a case analysis on α, β, γ, we give an example for each. For (1), when $(\alpha, \beta, \gamma) = (a, \{b\}, \{c\})$ we need $a \curlyvee b, c \prec b \vdash a \curlyvee c$ (which holds by right extension of $\curlyvee$). For (2) $(\alpha, \beta, \gamma) = (\{a\}, \{b\}, c)$ we need $b \prec a, c \prec b \vdash c \prec a$ (which holds by transitivity of $\prec$). Finally a counterexample for (3) is for $\alpha = a$, $\beta = \{b\}$ and $\gamma = c$, since $\alpha \prec \beta = a \curlyvee b$ and $\gamma \prec \beta = c \curlyvee b$ do not entail $\alpha \curlyvee \gamma = a \curlyvee c$ ($\curlyvee$ is not transitive). (See proof script [Mad14].) □

Corollary 4.5.19. *If $\alpha(x) \sqsubseteq_P \alpha_1(x)$ and $\alpha \curlyvee \beta$ is defined, then $\alpha_1 \curlyvee \beta$ is defined and $[\alpha \curlyvee \beta]\tau \sqsubseteq_P [\alpha_1 \curlyvee \beta]\tau$.*

Lemma 4.5.20. $\sqsubseteq_{P|Q} \subseteq (\sqsubseteq_P \cup \sqsubseteq_Q)^*$.

Proof. For visible prefixes, we conclude by transitivity of $\prec$ on extended names (Lemma 4.5.18). For conditional τ's, this is a consequence of Lemma 4.5.11. □

Lemma 4.5.21. *If $\mu \sqsubseteq_{P|Q} \mu'$ and $a \notin \mathrm{fn}(\mu), \mathrm{fn}(\mu'), \mathrm{fn}(Q)$ then $\mu \sqsubseteq_{(\boldsymbol{\nu} aP)|Q} \mu'$.*

Proof. We use Lemma 4.5.20 to obtain a sequence $\mu_0, \mu_1, \ldots, \mu_n$ with $\mu = \mu_0$ and $\mu' = \mu_n$ and $\mu_i \sqsubseteq^{\varphi_i} \mu_{i+1}$ with for each i, $P \rhd \varphi_i$ or $Q \rhd \varphi_i$. W.l.o.g. we can suppose when $a \in \mathrm{n}(\varphi_i)$ that $P \rhd \varphi_i$ (indeed, if instead we had $Q \rhd \varphi_i$, then φ_i would be trivial and thus $P \rhd \varphi_i$).

We then put together all subsequences $\varphi_i, \ldots, \varphi_j$ entailed by P, and do the same with Q, to obtain after some reindexing: $\mu_{j_1} \sqsubseteq_P \mu_{j_2} \sqsubseteq_Q \mu_{j_3} \sqsubseteq_P \ldots$ so that for all k, $a \notin \mathrm{fn}(\mu_{j_k})$. We can then write $\mu_{j_1} \sqsubseteq_{\boldsymbol{\nu} aP} \mu_{j_2} \sqsubseteq_Q \mu_{j_3} \sqsubseteq_{\boldsymbol{\nu} aP} \ldots$, and we are able to conclude that $\mu \sqsubseteq_{(\boldsymbol{\nu} aP)|Q} \mu$. □

Lemma 4.5.22. *If $\mu \sqsubseteq_{P|Q} \mu'$ and $a \notin \mathrm{fn}(\mu) \cup \mathrm{fn}(Q)$ then for some λ such that $a \notin \mathrm{fn}(\lambda)$, $\mu \sqsubseteq_{(\boldsymbol{\nu} aP)|Q} \lambda \sqsubseteq_P \mu'$.*

Proof. As in the proof of Lemma 4.5.21, we obtain $\mu = \mu_{j_1} \sqsubseteq_P \mu_{j_2} \sqsubseteq_Q \ldots \sqsubseteq_P \mu_{j_n} = \mu'$, except that a may appear in $\mu' = \mu_{j_n}$, but not in any other $\mu = \mu_{j_k}$ if $k < n$. We choose $\lambda = \mu_{j_{n-1}}$, the rest of the proof is similar. □

Lemma 4.5.23 is a tool to "shortcut" usages of preorder rules when reasoning by transition induction, to focus on the other rules. We use it frequently in the following, especially in Section 4.5.4.

Lemma 4.5.23. *If $P \xrightarrow{\mu} P'$ and $\eta \sqsubseteq_P \mu$ then $P \xrightarrow{\eta} P'$. Conversely, whenever $P \xrightarrow{\eta} P'$, there exists μ such that $\eta \sqsubseteq_P \mu$ and $P \xrightarrow{\mu} P'$, of which there is a proof, not bigger than the one for $P \xrightarrow{\eta} P'$, that does not end with a preorder rule.*

Proof. Both directions are proved by simple inductions. □

4.5.4 The characterisation theorem

Results about transitions.

Lemma 4.5.24. *If $(\boldsymbol{\nu} a)P \xrightarrow{\mu} P_1$ then $P_1 = (\boldsymbol{\nu} a)P'$ for some P' such that $P \xrightarrow{\mu} P'$.*

Proof. Using Lemma 4.5.23 we know that $(\boldsymbol{\nu} a)P \xrightarrow{\mu_1} P_1$ for some μ_1 such that $\mu \sqsubseteq_{\boldsymbol{\nu} aP} \mu_1$, and that this transition is coming from $P \xrightarrow{\mu_1} P'$ with $P_1 = (\boldsymbol{\nu} a)P'$. Since $\Phi(\boldsymbol{\nu} aP) \subseteq \Phi(P)$, we know that $\mu \sqsubseteq_P \mu_1$, so we can derive $P \xrightarrow{\mu} P'$. □

The proof of the next lemma, stating that transitions commute with $\equiv$, follows from an analysis similar to the proof of Lemma 4.5.15.

Notation We adopt the following notation in writing derivations, to denote either an application of the first part of Lemma 4.5.23 (i.e., an application of several preorder rules), or a recursive case analysis on the rules deriving $P \xrightarrow{\mu} P'$ until a non-preorder rule is reached, by application of the second part of Lemma 4.5.23.

$$\dfrac{\dfrac{P \xrightarrow{\mu'} P'}{[\mu \sqsubseteq_P \mu']}}{P \xrightarrow{\mu} P'}$$

Lemma 4.5.25. *If $P \equiv Q$ and $P \xrightarrow{\mu} P'$ then $Q \xrightarrow{\mu} \equiv P'$.*

Proof. More generally, we prove by induction on the derivation of $P \equiv Q$ that for all μ, (($P \xrightarrow{\mu} P'$ implies $Q \xrightarrow{\mu} \equiv P'$) and ($Q \xrightarrow{\mu} Q'$ implies $P \xrightarrow{\mu} \equiv Q'$)). Thanks to this formulation, precongruence is handled by transition induction and equivalence properties are trivial.

We give the remaining most complicated cases, one for associativity and one for extrusion. Then we perform an induction on $P \xrightarrow{\mu} P'$. We factor out the cases corresponding to preorder rules using Lemmas 4.5.23 and 4.5.15, and so we assume that the last rule is not a preorder rule.

1. $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$: we assume the last rule is $\rightarrow$-PAR-L. We use Lemma 4.5.23 to get $\mu \sqsubseteq_{P|Q} \mu_1$, then a $\rightarrow$-PAR-R rule to get $Q \xrightarrow{\mu_1} Q'$. Getting back $P \mid (Q \mid R) \xrightarrow{\mu_1} P \mid (Q' \mid R)$ is easy, what is more tricky is to get the label μ right: for that we use again Lemma 4.5.23 thanks to the fact that $\mu \sqsubseteq_{P|(Q|R)} \mu_1$, in turn implied by $\mu \sqsubseteq_{P|Q} \mu_1$ (since $\Phi(P \mid Q) \subseteq \Phi((P \mid Q) \mid R) = \Phi(P \mid (Q \mid R))$ by rule $\triangleright$-PAR-L and Lemma 4.5.15).

   We can sum up the reasoning above as follows:

$$\text{PAR-L}\ \dfrac{\dfrac{\dfrac{\text{PAR-R}\ \dfrac{Q \xrightarrow{\mu_1} Q'}{P \mid Q \xrightarrow{\mu_1} P \mid Q'}}{[\mu \sqsubseteq_{P|Q} \mu_1]}}{P \mid Q \xrightarrow{\mu} P \mid Q'}}{(P \mid Q) \mid R \xrightarrow{\mu} (P \mid Q') \mid R} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{Q \xrightarrow{\mu_1} Q'}{Q \mid R \xrightarrow{\mu_1} Q' \mid R}\ \text{PAR-L}}{P \mid (Q \mid R) \xrightarrow{\mu_1} P \mid (Q' \mid R)}\ \text{PAR-R}}{[\mu \sqsubseteq_{P|(Q|R)} \mu_1]}}{P \mid (Q \mid R) \xrightarrow{\mu} P \mid (Q' \mid R)}\ .$$

2. $\boldsymbol{\nu} a(P \mid Q) \equiv (\boldsymbol{\nu} aP) \mid Q$: any transition from $\boldsymbol{\nu} a(P \mid Q)$ must come from a transition from $P \mid Q$. After an application of Lemma 4.5.23 we have $\mu \sqsubseteq_{P|Q} \mu_1$ with a transition μ_1 that we can break into transitions from P and/or Q, depending on the last rule: we treat the case for $\rightarrow$-PAR-L and the case of $\rightarrow$-COM-* (the case for $\rightarrow$-PAR-R being simpler) and use Lemma 4.5.23 multiple times.

   (a) $\rightarrow$-PAR-L: action μ_1 (coming from P) may contain a. We use Lemma 4.5.22 to obtain a transition along λ ($a \notin \mathrm{fn}(\lambda)$), such that μ_1 can be transformed into λ (using only P) and then into μ (using both Q and $\boldsymbol{\nu} aP$). The following derivations illustrate the reasoning exposed above.

$$\dfrac{\dfrac{\dfrac{\dfrac{P \xrightarrow{\mu_1} P'}{P \mid Q \xrightarrow{\mu_1} P' \mid Q}}{[\mu \sqsubseteq_{P|Q} \mu_1]}}{P \mid Q \xrightarrow{\mu} P' \mid Q} \quad a \notin \mathrm{n}(\mu)}{(\boldsymbol{\nu} a)(P \mid Q) \xrightarrow{\mu} (\boldsymbol{\nu} a)(P' \mid Q)} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{P \xrightarrow{\mu_1} P'}{[\lambda \sqsubseteq_P \mu_1]}}{P \xrightarrow{\lambda} P'} \quad a \notin \mathrm{n}(\lambda)}{\boldsymbol{\nu} aP \xrightarrow{\lambda} \boldsymbol{\nu} aP'}}{(\boldsymbol{\nu} aP) \mid Q \xrightarrow{\lambda} (\boldsymbol{\nu} aP') \mid Q}}{[\mu \sqsubseteq_{(\boldsymbol{\nu} aP)|Q} \lambda]}}{(\boldsymbol{\nu} aP) \mid Q \xrightarrow{\mu} (\boldsymbol{\nu} aP') \mid Q}$$

(b) $\rightarrow$-COM-L: ($\rightarrow$-COM-R is symmetric) transition $\mu = [\varphi]\tau$ is obtained from transitions $\overline{\alpha}(x)$ and $\gamma(x)$ such that $[\varphi]\tau \sqsubseteq_{P|Q} [\alpha \curlyvee \gamma]\tau$. Since $a \notin \mathrm{fn}(Q)$ we already know that $\{a\} \neq \mathrm{n}(\gamma)$.
Suppose first that $\{a\} \neq \mathrm{n}(\alpha)$. Then we can obtain the transition $\overline{\alpha}$ from $(\boldsymbol{\nu} a)P$ as well, then the $[\alpha \curlyvee \gamma]\tau$ transition, and we conclude to get $[\varphi]\tau$ by Lemma 4.5.21.
The interesting case is when $\{a\} = \mathrm{n}(\alpha)$. We prove below the following implication:

$$[\varphi]\tau \sqsubseteq_{P|Q} [\alpha \curlyvee \gamma]\tau \implies \exists \beta \ \{a\} \neq \mathrm{n}(\beta) \wedge \begin{cases} [\varphi]\tau \sqsubseteq_{P|Q} [\beta \curlyvee \gamma]\tau \\ P \rhd \alpha \prec \beta \end{cases} \tag{4.4}$$

and we remark that (4.4) implies $\overline{\beta}(x) \sqsubseteq_P \overline{\alpha}(x)$. We can now use Lemma 4.5.23 to easily build back the $[\varphi]\tau$ transition from $(\boldsymbol{\nu} aP) \mid Q$, as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{P \xrightarrow{\overline{\alpha}(x)} P' \qquad Q \xrightarrow{\gamma(x)} Q'}{P \mid Q \xrightarrow{[\alpha \curlyvee \gamma]\tau} (\boldsymbol{\nu} x)(P' \mid Q')}}{\left[[\varphi]\tau \sqsubseteq_{P|Q} [\alpha \curlyvee \gamma]\tau\right]}}{P \mid Q \xrightarrow{[\varphi]\tau} (\boldsymbol{\nu} x)(P' \mid Q')} \quad a \notin \mathrm{n}(\varphi)}{(\boldsymbol{\nu} a)(P \mid Q) \xrightarrow{[\varphi]\tau} (\boldsymbol{\nu} a)(\boldsymbol{\nu} x)(P' \mid Q')} \rightsquigarrow \dfrac{\dfrac{\dfrac{\dfrac{P \xrightarrow{\overline{\alpha}(x)} P'}{\left[\overline{\beta}(x) \sqsubseteq_P \overline{\alpha}(x)\right]}}{P \xrightarrow{\overline{\beta}(x)} P' \quad a \notin \mathrm{n}(\beta)}}{\boldsymbol{\nu} aP \xrightarrow{\overline{\beta}(x)} \boldsymbol{\nu} aP' \qquad Q \xrightarrow{\gamma(x)} Q'}}{(\boldsymbol{\nu} aP) \mid Q \xrightarrow{[\beta \curlyvee \gamma]\tau} (\boldsymbol{\nu} x)((\boldsymbol{\nu} aP') \mid Q')}}{\left[[\varphi]\tau \sqsubseteq_{(\boldsymbol{\nu} aP)|Q} [\beta \curlyvee \gamma]\tau\right]}}{(\boldsymbol{\nu} aP) \mid Q \xrightarrow{[\varphi]\tau} (\boldsymbol{\nu} x)((\boldsymbol{\nu} aP') \mid Q')}$$

We prove now (4.4). Using Lemma 4.5.20 we decompose $[\varphi]\tau \sqsubseteq_{P|Q} [\alpha \curlyvee \gamma]\tau$ into a sequence of $\sqsubseteq^{\psi}$ with $P \rhd \psi \vee Q \rhd \psi$. We prove (4.4) by induction on this sequence, decomposing it "from the right" into $[\varphi]\tau \sqsubseteq_{P|Q} \eta \sqsubseteq^{\psi} [\alpha \curlyvee \gamma]\tau$: we assume (4.4) holds when $[\alpha \curlyvee \gamma]\tau$ is replaced with η and we decompose $\eta \sqsubseteq^{\psi} [\alpha \curlyvee \gamma]\tau$ into easy cases:

- $\eta = [\beta \curlyvee \gamma]\tau$ and $\psi = \alpha \prec \beta$ and $\mathrm{n}(\alpha) \neq \mathrm{n}(\beta)$:
  in this case, $P \rhd \psi$ and since $[\varphi]\tau \sqsubseteq_{P|Q} [\beta \curlyvee \gamma]\tau$ we are done.
- $\eta = [\beta \curlyvee \gamma]\tau$ and $\psi = \alpha \prec \beta$ and $\mathrm{n}(\alpha) = \mathrm{n}(\beta)$:
  in this case, $P \rhd \psi$ (even if $Q \rhd \psi$). We use the induction hypothesis to get β' such that $[\varphi]\tau \sqsubseteq_{P|Q} [\beta' \curlyvee \gamma]\tau$ and $\mathrm{n}(\beta') \neq \mathrm{n}(\beta)$ and $P \rhd \beta \prec \beta'$. This β' is enough, since $P \rhd \alpha \prec \beta'$.
- $\eta = [\alpha \curlyvee \delta]\tau$ with $\mathrm{n}(\delta) = \{a\}$:
  we interrupt the induction since we can derive a plain τ, hence any $[\varphi]\tau$ that we want.
- $\eta = [\alpha \curlyvee \delta]\tau$ and $\psi = \gamma \prec \delta$ and $P \rhd \psi$ or $Q \rhd \psi$:
  we use the induction hypothesis to get the expected β and we replay ψ on top of it (the induction hypothesis gives $[\varphi]\tau \sqsubseteq_{P|Q} [\beta \curlyvee \delta]\tau$, from which through $P, Q \rhd \gamma \prec \delta$ it is easy to get $[\varphi]\tau \sqsubseteq_{P|Q} [\beta \curlyvee \gamma]\tau$).

This concludes the case analysis.

□

Lemma 4.5.26. *$\equiv$ is a bisimulation.*

Proof. This follows from Lemmas 4.5.15 and 4.5.25. The correspondence between transitions is exact, hence the only non-trivial case is for the $[\varphi]\tau$ transition because one requires $P' \mid \varphi \equiv Q' \mid \varphi$ instead of just $P' \equiv Q'$. This holds because $\equiv$ is a congruence, by definition. □

We now move on to congruence results. To derive those, we use several up-to techniques to reduce the size of the bisimulation candidates and factor out uninteresting case analyses. Note that we develop up-to techniques in much more details in Chapter 5, which includes a gentle introduction (Section 5.1).

Definition 4.5.27 (Bisimulation up to $\sim$)**.** A relation $\mathcal{R}$ is a bisimulation up to bisimilarity if it validates the clauses in the definition of $\sim$ (Definition 4.5.5), except that we require $P' \sim\mathcal{R}\sim Q'$ instead of $P' \mathcal{R} Q'$ and $(P' \mid \varphi) \sim\mathcal{R}\sim (Q' \mid \varphi)$ instead of $(P' \mid \varphi) \mathcal{R} (Q' \mid \varphi)$.

Lemma 4.5.28. *If $\mathcal{R}$ is a bisimulation up to bisimilarity, then $\mathcal{R} \subseteq \sim$.*

Proof. We prove $\sim\mathcal{R}\sim$ is a bisimulation. The only unusual transition μ is when $\mu = [\varphi]\tau$, but from $P \sim P_1 \mathcal{R} Q_1 \sim Q$ and $P \xrightarrow{\mu} P'$ we know:

- that $P_1 \xrightarrow{\mu} P_1'$ and $(P' \mid \varphi) \sim (P_1' \mid \varphi)$ from the $\sim$ game,
- that $Q_1 \xrightarrow{\mu} Q_1'$ and $(P_1' \mid \varphi) \sim\mathcal{R}\sim (Q_1' \mid \varphi)$ from the hypothesis on $\mathcal{R}$,
- that $P \xrightarrow{\mu} P'$ and $(Q_1' \mid \varphi) \sim (Q' \mid \varphi)$ from the second $\sim$ game.

We conclude by transitivity of $\sim$ since $(P' \mid \varphi) \sim\sim\mathcal{R}\sim\sim (Q' \mid \varphi)$. □

Lemma 4.5.29. *If $P \rhd \varphi$ then $P \mid \varphi \sim P$.*

Proof. We prove that $\mathcal{R} = \{(P \mid \varphi, P) \mid P \rhd \varphi\}$ is a bisimulation up to $\equiv$ (and thus is included in $\sim$ by Lemmas 4.5.28 and 4.5.26). First, all transitions from P can be done by $P \mid \varphi$; sometimes we have to relate $P \mid \psi$ to $(P \mid \varphi) \mid \psi$ which we rewrite as $(P \mid \psi) \mid \varphi$ using $\equiv$ (and indeed $(P \mid \psi) \mid \varphi \mathcal{R} (P \mid \psi)$).

Second, suppose $P \mid \varphi \xrightarrow{\mu} P_1$. Using Lemma 4.5.23 we get μ_1 such that $\mu \sqsubseteq_{P\mid\varphi} \mu_1$, and a proof of $P \mid \varphi \xrightarrow{\mu_1} P'$ with a non preorder rule as last rule. This rule must be a PAR rule coming from P so in fact $P_1 = P' \mid \varphi$ with $P \xrightarrow{\mu_1} P'$. Since $\Phi(P \mid \varphi) = \Phi(P)$ by Lemma 4.5.14, we know that $\mu \sqsubseteq_P \mu_1$. We can apply Lemma 4.5.23 again to deduce $P \xrightarrow{\mu} P'$, and indeed $P' \mid \varphi \mathcal{R} P'$. □

Lemma 4.5.30. *$P \sim Q$ implies $(\boldsymbol{\nu} a)P \sim (\boldsymbol{\nu} a)Q$ for all a.*

Proof. We prove $\mathcal{R} \triangleq \{((\boldsymbol{\nu} a)P, (\boldsymbol{\nu} a)Q) \mid P \sim Q\}$ is a bisimulation up to bisimilarity (Lemma 4.5.28), which is relatively easy using Lemma 4.5.24. The only interesting case is for a conditional τ transition. If $(\boldsymbol{\nu} a)P \xrightarrow{[\varphi]\tau} (\boldsymbol{\nu} a)P'$ then $P \xrightarrow{[\varphi]\tau} P'$ and using $\sim$, $Q \xrightarrow{[\varphi]\tau} Q'$ with $(P' \mid \varphi) \sim (Q' \mid \varphi)$. Since we want to relate $(\boldsymbol{\nu} a)P' \mid \varphi$ and $(\boldsymbol{\nu} a)Q' \mid \varphi$ we $\sim$-rewrite them into $(\boldsymbol{\nu} a)(P' \mid \varphi)$ and $(\boldsymbol{\nu} a)(Q' \mid \varphi)$ (using Lemma 4.5.26) which are indeed related through $\mathcal{R}$. The condition about $\rhd$ is ensured by compositionality of $\rhd$ (Lemma 4.5.13). □

Definition 4.5.31 (Bisimulation up to $\sim$ and $\boldsymbol{\nu}$)**.** A relation $\mathcal{R}$ is a bisimulation up to restriction and bisimilarity if it validates the clauses of the usual bisimulation, except that the outcomes of the transitions, P_1 and Q_1, are requested to satisfy $P_1 \sim (\boldsymbol{\nu}\widetilde{a})P_2$ and $Q_1 \sim (\boldsymbol{\nu}\widetilde{a})Q_2$ with $P_2 \mathcal{R} Q_2$, where $\widetilde{a}$ stands for a (possibly empty) tuple of names.

Lemma 4.5.32. *If $\mathcal{R}$ is a bisimulation up to restriction and bisimilarity then $\mathcal{R} \subseteq \sim$.*

Proof. We write $\mathcal{R}^\nu$ for $\{(\boldsymbol{\nu}\tilde{a})P, (\boldsymbol{\nu}\tilde{a})Q) \mid P \mathcal{R} Q\}$. We know that $\mathcal{R} \subseteq \mathcal{S} \triangleq \sim \mathcal{R}^\nu \sim$ so it is enough to prove that $\mathcal{S}$ is a bisimulation. Suppose $P \sim (\boldsymbol{\nu}\tilde{a})P_1 \ \mathcal{R}^\nu \ (\boldsymbol{\nu}\tilde{a})Q_1 \sim Q$ with $P_1 \mathcal{R} Q_1$. We start from a transition $P \xrightarrow{\mu} P'$, we use the bisimulation game on $\sim$ and Lemma 4.5.24 to deduce $(\boldsymbol{\nu}\tilde{a})P_1 \xrightarrow{\mu} (\boldsymbol{\nu}\tilde{a})P_1'$ with $P_1 \xrightarrow{\mu} P_1'$ and $(P' \mid A) \sim ((\boldsymbol{\nu}\tilde{a})P_1' \mid A)$ with $A = \varphi$ if $\mu = [\varphi]\tau$ or $A = 0$ otherwise. Using the bisimulation up to on $\mathcal{R}$ we obtain $Q_1 \xrightarrow{\mu} Q_1'$ with $(P_1' \mid A) \sim (\boldsymbol{\nu}\tilde{c})P_2 \ \mathcal{R}^\nu \ (\boldsymbol{\nu}\tilde{c})Q_2 \sim (Q_1' \mid A)$ for some $P_2 \mathcal{R} Q_2$. We also get $(\boldsymbol{\nu}\tilde{a})Q_1 \xrightarrow{\mu} (\boldsymbol{\nu}\tilde{a})Q_1'$ from which using the game on the second $\sim$, $Q \xrightarrow{\mu} Q'$ with $((\boldsymbol{\nu}\tilde{a})Q_1' \mid A) \sim (Q' \mid A)$.

We now compose what we have: $(P' \mid A) \sim ((\boldsymbol{\nu}\tilde{a})P_1' \mid A)$ and $(P_1' \mid A) \sim ((\boldsymbol{\nu}\tilde{c})P_2)$. We can get from the latter $((\boldsymbol{\nu}\tilde{a})P_1' \mid A) \sim ((\boldsymbol{\nu}\tilde{a}\tilde{c})P_2)$ using Lemmas 4.5.26 and 4.5.30, and then compose them using transitivity of $\sim$; similarly for Q: $(Q' \mid A) \sim ((\boldsymbol{\nu}\tilde{a}\tilde{c})Q_2)$. Since $P_2 \ \mathcal{R} \ Q_2$, the pair $(P' \mid A, Q' \mid A)$ is in $\mathcal{S}$. □

Definition 4.5.33 (Bisimulation up to $\sim$ and σ)**.** A relation $\mathcal{R}$ is a bisimulation up to bisimilarity and injective substitution if it validates the clauses of the usual bisimulation, except that the outcomes of the transitions, P_1 and Q_1, are requested to satisfy $P_1 \sim P_{2_\sigma}$ and $Q_1 \sim Q_{2_\sigma}$ with $P_2 \mathcal{R} Q_2$, where σ stands for an injective name substitution.

Proof. We prove $\sim\mathcal{R}_\sigma\sim$ is a bisimulation where $\mathcal{R}_\sigma$ stands for $\{(P_\sigma, Q_\sigma) \mid P \mathcal{R} Q \text{ and } \sigma \text{ is injective}\}$. □

Lemma 4.5.34 is the most complex case in the proof of congruence of $\sim$.

Lemma 4.5.34. *$P \sim Q$ implies $P \mid R \sim Q \mid R$ for all R.*

Proof. We prove that $\mathcal{R} = \{(P \mid R, Q \mid R) \mid P \sim Q\}$ is a bisimulation up to restriction and bisimilarity. Lemma 4.5.13 handles the first clause about conditions, and Lemma 4.5.23 the second one about visible transitions. Lemma 4.5.23 also helps handling a $[\varphi]\tau$ transition from $P \mid R$ to P_1 by saying it comes from a $[\varphi_1]\tau$ transition from P, or R, or both, and such that $[\varphi]\tau \sqsubseteq_{P|R} [\varphi_1]\tau$. From that, getting a transition $[\varphi]\tau$ from $Q \mid R$ to Q_1 is easy, but one must relate $P_1 \mid \varphi$ and $Q_1 \mid \varphi$ in $\mathcal{R}$ using three different assumptions:

1. $R \xrightarrow{[\varphi_1]\tau} R'$. Then $P \mid R' \mid \varphi \sim\mathcal{R}\sim Q \mid R' \mid \varphi$. (up to $\sim$)
2. $P \xrightarrow{[\varphi_1]\tau} P'$. Then $P' \mid \varphi_1 \sim Q' \mid \varphi_1$, which implies $P' \mid R \mid \varphi \sim\mathcal{R}\sim Q' \mid R \mid \varphi$. Indeed φ_1 is absorbed by φ since $P \mid Q \mid \varphi \rhd \varphi_1$. (up to $\sim$)
3. P and R synchronise into $(\boldsymbol{\nu} x)(P' \mid R')$. Then $P' \sim Q'$ and we can relate $(\boldsymbol{\nu} x)(P' \mid R') \mid \varphi$ to $(\boldsymbol{\nu} x)(Q' \mid R') \mid \varphi$ (up to $\sim$ and $\boldsymbol{\nu}$).

□

Proposition 4.5.35 (Congruence of $\sim$)**.** *Bisimilarity is a congruence.*

Proof. The difficult case is closure under parallel composition (Lemma 4.5.34). Closure under restrictions is proven in Lemma 4.5.30, and closure under guarded sums is easy but still need Lemmas 4.5.34 and 4.5.30, since arcs and restrictions can appear in the rules for the prefixes and the bisimulation clause for $[\varphi]\tau$. □

The following lemma is useful for the proof of completeness of $\sim$ w.r.t. $\simeq$.

Lemma 4.5.36. *For any P, Q and φ, $P \xrightarrow{[\varphi]\tau} Q$ iff $P \mid \varphi \xrightarrow{\tau} Q \mid \varphi$.*

Proof. If $P \xrightarrow{[\varphi]\tau} P'$, then $P \mid \varphi \xrightarrow{[\varphi]\tau} P' \mid \varphi$ and $P \mid \varphi \xrightarrow{\tau} P' \mid \varphi$ by Lemma 4.5.23, since $\tau \sqsubseteq^\varphi [\varphi]\tau$ and $P \mid \varphi \rhd \varphi$.

From $P \mid \varphi \xrightarrow{\tau} P' \mid \varphi$ Lemma 4.5.23 provides a condition ζ such that $P \xrightarrow{[\zeta]\tau} P'$ and $\tau \sqsubseteq_{P|\varphi} [\zeta]\tau$, which implies in turn $P \mid \varphi \rhd \zeta$. We prove by induction on $P \mid \varphi \rhd \zeta$ that $[\varphi]\tau \sqsubseteq_P [\zeta]\tau$ (see proof script in [Mad14]) and conclude with Lemma 4.5.23. □

The following "Harmony lemma" relates reduction and $\xrightarrow{\tau}$ transitions.

Lemma 4.5.37. *If $P \longrightarrow P'$ then $P \xrightarrow{\tau}\sim P'$, and if $P \xrightarrow{\tau} P'$ then $P \longrightarrow\sim P'$.*

Proof. Follows from Lemma 4.5.25 ($\longrightarrow$ produces an arc a/b when $\xrightarrow{\tau}$ produces the bisimilar process $(\boldsymbol{\nu} x)(a/x \mid x/b)$). □

Theorem 4.5.38 (Characterisation)**.** *$P \simeq Q$ iff $P \sim Q$.*

Proof. The proof follows a standard pattern. Soundness is a consequence of congruence (Proposition 4.5.35), Lemma 4.5.37, and the correspondence of barbs with visible transitions (which follows from Lemma 4.5.25).

For completeness, we have to show that contexts can express the conditions in the three clauses in Definition 4.5.5. We define accordingly tester processes. The first clause is handled using process $[\varphi]\tau.\overline{w}$. We use φ for the second clause (by Lemma 4.5.36). For transitions (third clause), the counterpart of, e.g., $\xrightarrow{\{a\}(x)}$, is given by tester process $\overline{a}(y).(z/y \mid \overline{w} \mid w)$. □

4.5.5 Correspondences between variants of the calculus

The calculus of the previous sections is a version of $\pi\mathrm{P}^+$ with prefixes for free input and output, and without the corresponding bound prefixes (as well as without sum and conditional τ). That calculus ($\pi\mathrm{P}$) is equipped with the equivalence $\simeq_{\mathrm{bn}}$ from Definition 4.3.7. The encoding $[\cdot]_{\mathrm{f}}$, which we introduced in Remark 4.5.4, allows us to embed $\pi\mathrm{P}$ into $\pi\mathrm{P}^+$ in a faithful way:

Proposition 4.5.39 (Correspondence with Definition 4.3.7)**.** $P \simeq_{\mathrm{bn}} Q$ *iff* $[P]_{\mathrm{f}} \simeq [Q]_{\mathrm{f}}$.

Proof. $[P]_{\mathrm{f}} \simeq [Q]_{\mathrm{f}} \Rightarrow P \simeq_{\mathrm{bn}} Q$ follows from the fact that $[\cdot]_{\mathrm{f}}$ induces standard correspondences of reductions, barbs, and contexts, from $\pi\mathrm{P}$ to $\pi\mathrm{P}^+$:

- if $P \longrightarrow_{\mathrm{bn}} P'$ then $[P]_{\mathrm{f}} \longrightarrow\simeq [P']_{\mathrm{f}}$,
- if $[P]_{\mathrm{f}} \longrightarrow P_1$ then $P_1 \simeq [P']_{\mathrm{f}}$ and $P \longrightarrow_{\mathrm{bn}} P'$ for some P',
- $P \downarrow_a^{\mathrm{bn}}$ iff $[P]_{\mathrm{f}} \downarrow_a$
- if C is a $\pi\mathrm{P}$ context, for some $\pi\mathrm{P}^+$ context D, for all P, $[C[P]]_{\mathrm{f}} = D[[P]_{\mathrm{f}}]$.

Consider now $\pi\mathrm{P}^-$, the subcalculus of $\pi\mathrm{P}^+$ with neither sum nor $[\varphi]\tau$ constructs, and $\simeq^-$, the barbed congruence induced by $\pi\mathrm{P}^-$ contexts. The proof of completeness of $\sim$ only uses $\pi\mathrm{P}^-$ contexts, hence $\simeq^- = \sim = \simeq$.

We prove the above correspondences between $\pi\mathrm{P}^-$ and $\pi\mathrm{P}$ for the encoding $[P]_{\mathrm{p}}^{\mathrm{b}} \triangleq [[P]_{\mathrm{p}}]^{\mathrm{b}}$ where $[\cdot]^{\mathrm{b}}$ is the standard encoding of bound prefixes using free prefixes (e.g., $[a(x).P]^{\mathrm{b}} \triangleq (\boldsymbol{\nu} x)ax.\,[P]^{\mathrm{b}}$). This implies that $[P]_{\mathrm{p}}^{\mathrm{b}} \simeq_{\mathrm{bn}} [Q]_{\mathrm{p}}^{\mathrm{b}} \Rightarrow P \simeq^- Q$ for two $\pi\mathrm{P}^-$ processes P and Q.

Remarking that $[\cdot]_{\mathrm{f}} : \pi\mathrm{P} \to \pi\mathrm{P}^-$, we compose the encodings and we prove $P \simeq_{\mathrm{bn}} [[P]_{\mathrm{f}}]_{\mathrm{p}}^{\mathrm{b}}$ by induction on P using the following $\pi\mathrm{P}$ laws (when x is fresh):

$$\overline{a}b.P \simeq_{\mathrm{bn}} (\boldsymbol{\nu} x)\overline{a}x.(b/x \mid P) \qquad ab.P \simeq_{\mathrm{bn}} (\boldsymbol{\nu} x)ax.(x/b \mid P) \ .$$

Finally, if $P \simeq_{\mathrm{bn}} Q$, then $[[P]_{\mathrm{f}}]_{\mathrm{p}}^{\mathrm{b}} \simeq_{\mathrm{bn}} [[Q]_{\mathrm{f}}]_{\mathrm{p}}^{\mathrm{b}}$, and thus $[P]_{\mathrm{f}} \simeq [Q]_{\mathrm{f}}$. □

4.5.6 Labelled transitions for free prefixes

We now consider $\pi\mathrm{P}_{\mathrm{F}}^+$, the variant of $\pi\mathrm{P}$ where only the grammar for prefixes of is changed, as follows ($\pi\mathrm{P}_{\mathrm{F}}^+$ is $\pi\mathrm{P}$ with sums and conditional τ):

$$\pi \ ::= \ \alpha b \ \mid \ \overline{\alpha} b \ \mid \ [\varphi]\tau$$

We adapt the LTS of Section 4.5.2 to $\pi\mathrm{P}_{\mathrm{F}}^+$. The only modification we need to make is to replace the rules for bound prefixes by the following:

$$\frac{x \notin \mathrm{n}(\alpha) \cup \{b\} \cup \mathrm{fn}(P)}{\alpha b.P \xrightarrow{\alpha(x)} x/b \mid P} \ {\scriptstyle \rightarrow\text{-F-IN}} \qquad \frac{x \notin \mathrm{n}(\alpha) \cup \{b\} \cup \mathrm{fn}(P)}{\overline{\alpha} b.P \xrightarrow{\overline{\alpha}(x)} b/x \mid P} \ {\scriptstyle \rightarrow\text{-F-OUT}}$$

The calculi $\pi\mathrm{P}_{\mathrm{F}}^+$ and $\pi\mathrm{P}^+$ are similar, and can be encoded into each other. The domain of $[\cdot]_{\mathrm{f}}$ can be trivially extended to $\pi\mathrm{P}_{\mathrm{F}}^+$ (extended names are handled like regular ones, e.g. $[\alpha b.P]_{\mathrm{f}} \triangleq \alpha(x).(x/b \mid [P]_{\mathrm{f}})$). In the other direction, we need to extend $[\cdot]^{\mathrm{b}}$ to $\pi\mathrm{P}^+$, with some care to handle binders in sums:

$$\left[\textstyle\sum_i \pi_i.P_i\right]^{\mathrm{b}} \ \triangleq \ (\boldsymbol{\nu} x) \sum_i \begin{cases} \overline{\alpha} x.P_i & \text{if} \quad \pi_i = \overline{\alpha}(x) \\ \alpha x.P_i & \text{if} \quad \pi_i = \alpha(x) \\ [\varphi]\tau.P_i & \text{if} \quad \pi_i = [\varphi]\tau \end{cases} ,$$

where x is chosen fresh in the third clause. Note that we rely on α-conversion to have the same name x bound in all summands in the encoding of sums.

We can show that both encodings $[\cdot]_{\mathrm{f}}$ and $[\cdot]^{\mathrm{b}}$ preserve transitions and bisimilarity, so, as in Proposition 4.5.39, they preserve barbed congruence.

4.6 Axiomatisation

In this section, we provide a second characterisation of barbed congruence, by presenting a set of laws that define an axiomatisation of $\simeq$. Algebraic laws help analysing the behaviour of the constructs of the calculus and their interplay. We present a sample of behavioural equalities, and explain how they can be derived equationally, in Section 4.6.1 (page 71).

The axiomatisation we give is less simple than, say, the one for fusions in [PV98a], for two reasons: first, we manipulate preorders between names rather than equivalence relations, the fact that Υ is not transitive making the problem more intricate. Second, the preorder is explicitly represented in processes, so that some equational laws must describe the interplay between processes and the preorder relation. On the contrary, such aspects are dealt with implicitly in fusions. We show how our ideas can be adapted to provide an axiomatisation of behavioural equivalence for explicit fusions in Section 4.6.3.

The axiomatisation exploits the idea that $\pi\mathrm{P}^+$ processes have a *state* component, corresponding to the preorder induced by arcs. Several laws in the axiomatisation express persistence of the state component (the state can only be extended along computation, since arcs are persistent in $\pi\mathrm{P}^+$). Moreover, the restriction operator prevents the state from being globally shared in general: for instance, in process P_0 in (4.3) (page 56), name a can be used instead of c, but is only known inside the scope of $(\boldsymbol{\nu} a)$. All in all, the handling of restriction in our axiomatisation requires more care than is usually the case, due to the necessity to express the "view" that subprocesses have on the preorder of names.

4.6.1 Equational laws for strong bisimilarity

Notations and terminology. We use A to range over processes that consist of compositions of φ processes only, which we call *preorder processes*. We often view such processes as multisets of conditions. We use notation A, P to denote a process that can be written, *using the monoid laws for parallel composition*, as $A \mid P$, where P does not contain toplevel arcs. Note that A may contain restrictions, corresponding to the definition of join processes (given in Section 4.5.1), but extrusion is *not* allowed to decompose a process as A, P.

We write $\vdash P = Q$ whenever P and Q can be related by equational reasoning using the laws of Table 4.2, together with the following standard axiom:

Expansion law (we can suppose $x \neq y$, $\mathrm{bn}(\pi_i) \notin \mathrm{fn}(T)$, $\mathrm{bn}(\rho_j) \notin \mathrm{fn}(S)$.)

$$\underbrace{\sum_i \pi_i.P_i}_{S} \mid \underbrace{\sum_j \rho_j.R_j}_{T} = \sum_i \pi_i.(P_i \mid T) + \sum_j \rho_j.(S \mid R_j) \quad \text{when } \alpha \,\Upsilon\, \beta \text{ is defined.}$$

$$+ \sum_{i,j} [\alpha \,\Upsilon\, \beta]\tau.(\boldsymbol{\nu} xy)(x/y \mid P_i \mid R_j) \quad \text{and } \{\pi_i, \rho_j\} = \{\overline{\alpha}(x), \beta(y)\}$$

Note that we omit the standard laws expressing that $\mid$ and $+$ obey the laws of commutative monoids, and that $+$ is idempotent. We also omit the laws for equational reasoning (equivalence, substitutivity). We will reason up to these laws in the remainder.

Comments on the laws. Before presenting the properties of the axiomatisation, we comment on the laws of our axiomatisation and illustrate them on some examples.

As usual, the expansion law allows us to rewrite the parallel composition of two sums into a sum, the third summand describing synchronisation in $\pi\mathrm{P}^+$.

Preorders. Laws L1-L4 express basic properties of relations $\prec$ and Υ, and actually provide an axiomatisation of $\sim$ for preorder processes.

Laws for preorder processes

$$\text{L1}\quad a \prec b \mid b \prec c \;=\; a \prec b \mid b \prec c \mid a \prec c \qquad \text{L2}\quad a \prec b \mid c \prec b \;=\; a \prec b \mid c \prec b \mid a \curlyvee c$$
$$\text{L3}\quad a \prec b \mid b \curlyvee c \;=\; a \prec b \mid b \curlyvee c \mid a \curlyvee c \qquad \text{L4}\quad a \prec a = 0$$

Laws for prefixes (the counterparts of laws L10-L12 for output are omitted)

$$\text{L5}\quad \varphi, S + \pi.P = \varphi, S + \pi.(\varphi \mid P)$$
$$\text{L6}\quad [\varphi]\tau.P = [\varphi]\tau.(\varphi \mid P)$$
$$\text{L7}\quad [a \prec a]\tau.P = [b \curlyvee b]\tau.P$$
$$\text{L8}\quad [a \curlyvee b]\tau.P = [a \curlyvee b]\tau.P + [a \prec b]\tau.P$$
$$\text{L9}\quad [a \curlyvee b]\tau.P = [a \curlyvee b]\tau.P + [b \curlyvee a]\tau.P$$

$$\text{L10}\quad a(x).P = a(x).P + \{a\}(x).P$$
$$\text{L11}\quad b/a, S + a(x).P = b/a, S + a(x).P + b(x).P$$
$$\text{L12}\quad a/b, S + \{a\}(x).P = a/b, S + \{a\}(x).P + \{b\}(x).P$$

$$\text{L13}\quad b/a, S + [a \prec c]\tau.P = b/a, S + [a \prec c]\tau.P + [b \prec c]\tau.P$$
$$\text{L14}\quad a/b, S + [c \prec a]\tau.P = a/b, S + [c \prec a]\tau.P + [c \prec b]\tau.P$$
$$\text{L15}\quad b/a, S + [a \curlyvee c]\tau.P = b/a, S + [a \curlyvee c]\tau.P + [b \curlyvee c]\tau.P$$
$$\text{L16}\quad b/a, S + [a \curlyvee c]\tau.P = b/a, S + [a \curlyvee c]\tau.P + [c \prec b]\tau.P$$

$$\text{L17}\quad \alpha(y).P = \alpha(x).(\boldsymbol{\nu} y)(x/y \mid P) \quad \text{if } x \notin \mathrm{fn}(P)$$
$$\text{L18}\quad \overline{\alpha}(y).P = \overline{\alpha}(x).(\boldsymbol{\nu} y)(y/x \mid P) \quad \text{if } x \notin \mathrm{fn}(P)$$

Laws for restriction (the counterparts of laws L25 and L26 for output are omitted; $a \prec b \in A^{\neq}$ stands for $a \prec b \in A$ and $a \neq b$, and similarly for $a \curlyvee b$.)

$$\text{L19}\quad (\boldsymbol{\nu} b)P = (\boldsymbol{\nu} a)(P\{a/b\}) \quad \text{if } a \notin \mathrm{fn}(P)$$
$$\text{L20}\quad P \mid (\boldsymbol{\nu} a)Q = (\boldsymbol{\nu} a)(P \mid Q) \quad \text{if } a \notin \mathrm{fn}(P)$$
$$\text{L21}\quad (\boldsymbol{\nu} c)(\boldsymbol{\nu} d)P = (\boldsymbol{\nu} d)(\boldsymbol{\nu} c)P$$
$$\text{L22}\quad (\boldsymbol{\nu} a)0 = 0$$

$$\text{L23}\quad (\boldsymbol{\nu} a)A = \{b \prec c \mid b \prec a,\ a \prec c \in A^{\neq}\} \uplus \{b \curlyvee c \mid b \prec a,\ c \prec a \in A^{\neq}\} \uplus \{b \curlyvee c \mid a \curlyvee c,\ b \prec a \in A^{\neq}\} \uplus \{\varphi \in A \mid a \notin \mathrm{n}(\varphi)\}$$

$$\text{L24}\quad (\boldsymbol{\nu} a)(A,\ S + \pi.P) = (\boldsymbol{\nu} a)\big(A,\ S + \pi.(\boldsymbol{\nu} a)(A \mid P)\big) \qquad a \notin \mathrm{n}(\pi)$$

$$\text{L25}\quad (\boldsymbol{\nu} a)(A,\ S + a(x).P) = (\boldsymbol{\nu} a)\Big(A,\ S + \textstyle\sum_{a \prec b \in A^{\neq}} b(x).(\boldsymbol{\nu} a)(A \mid P) + \sum_{\substack{b \prec a \in A^{\neq} \\ \vee\, a \curlyvee b \in A^{\neq}}} \{b\}(x).(\boldsymbol{\nu} a)(A \mid P)\Big)$$

$$\text{L26}\quad (\boldsymbol{\nu} a)(A,\ S + \{a\}(x).P) = (\boldsymbol{\nu} a)\Big(A,\ S + \textstyle\sum_{b \prec a \in A^{\neq}} \{b\}(x).(\boldsymbol{\nu} a)(A \mid P)\Big)$$

$$\text{L27}\quad (\boldsymbol{\nu} a)(A,\ S + [a \prec c]\tau.P) = (\boldsymbol{\nu} a)\Big(A,\ S + \textstyle\sum_{a \prec b \in A^{\neq}} [b \prec c]\tau.(\boldsymbol{\nu} a)(A \mid P)\Big) \qquad a \neq c$$

$$\text{L28}\quad (\boldsymbol{\nu} a)(A,\ S + [c \prec a]\tau.P) = (\boldsymbol{\nu} a)\Big(A,\ S + \textstyle\sum_{b \prec a \in A^{\neq}} [c \prec b]\tau.(\boldsymbol{\nu} a)(A \mid P)\Big) \qquad a \neq c$$

$$\text{L29}\quad (\boldsymbol{\nu} a)(A,\ S + [a \curlyvee c]\tau.P) = (\boldsymbol{\nu} a)\Big(A,\ S + \textstyle\sum_{a \prec b \in A^{\neq}} [b \curlyvee c]\tau.(\boldsymbol{\nu} a)(A \mid P) + \sum_{\substack{b \prec a \in A^{\neq} \\ \vee\, a \curlyvee b \in A^{\neq}}} [c \prec b]\tau.(\boldsymbol{\nu} a)(A \mid P)\Big) \qquad a \neq c$$

Table 4.2: An axiomatisation of $\sim$

Prefixes. Law L5 propagates φs in depth, expressing the persistence of condition processes (φ). Law L6 is the counterpart of the third clause of Definition 4.5.5, and describes the outcome of a $[\varphi]\tau$ transition. Similarly, laws L17-L18 correspond to the firing of visible transitions in the LTS (regarding these rules, see also the comments after Proposition 4.6.5).

We note that α-conversion for input prefixes follows from laws L17 and L19, by deriving the following equalities (and similarly for the other visible prefixes):

$$a(y).P \overset{L17}{=} a(x).(\boldsymbol{\nu} y)(x/y \mid P) \overset{L19}{=} a(x).(\boldsymbol{\nu} y')(x/y' \mid P\{y'/y\}) \overset{L17}{=} a(y').P\{y'/y\}\,.$$

Laws L11-L16 can be used to expand process behaviours using the preorder: arcs can modify the subject of visible prefixes (L11-L12) and the condition in $[\varphi]\tau$ prefixes (L13-L16). Laws L8, L9 and L13-L16 rely on the defining properties of relations $\prec$ and Υ. Finally, law L7 is used to equate all reflexive τ prefixes.

Restriction. Laws L19-L22 are standard. The other laws are used to "push" restrictions inside processes. Due to the necessity to handle the preorder component (A), they are rather complex.

Law L23 is used to eliminate a restriction on a name a in a preorder process, by propagating the information expressed by all φs that mention a.

Law L24 is rather self-explanatory, and shows how the A component prevents us from simply pushing the restriction downwards (under prefixes).

Laws L25-L29 describe a kind of "synchronous application" of the prefix laws seen above. For instance, the two summands in law L25 correspond to applications of laws L11-L12: as we push the restriction on a downwards, we make sure that all possible applications of these laws are taken into account.

Intuitively, L23 is used after laws L24-L29 have been used to erase all prefixes mentioning the restricted name a, pushing the restriction on a inwards.

All in all, the set of laws in Table 4.2 is rather lengthy. We make two comments on this. First, it can be remarked that axiomatisations often treat restriction separately, by first focusing on a restriction-free calculus. In $\pi\mathrm{P}^+$, because of preorder processes, we cannot in general push restrictions on top of sum processes, so the situation is more complex.

Second, we could have presented the laws in a more compact way, by writing *schemas*. A uniform presentation for laws L7-L16 and L25-L29 is as follows:

$$\frac{\eta \sqsubseteq_A \mu \quad \mu.P \in S}{A, S = A, S + \eta.P} \qquad \frac{a \in \mathrm{fn}(\mu) \quad \forall \eta \sqsubseteq_A \mu \quad a \in \mathrm{fn}(\eta) \vee \exists \rho \;\; \eta \sqsubseteq_A \rho \;\wedge\; \rho.P \in S}{(\boldsymbol{\nu} a)(A, \mu.P + S) = (\boldsymbol{\nu} a)(A, S)}$$

(To remove $\mu.P$ from $\mu.P + S$, the second rule requires that some $\rho.P$ are in S. The second rule can be used to add those summands to S.) We prefer however to write all rules explicitly, since this is how they are handled in proofs.

Examples of derivable equalities. In the following examples, we sometimes switch silently to notation A, P to ease readability. We also allow ourselves to simplify some reasonings involving prefixes where the object is not important. We explain how the following equalities between $\pi\mathrm{P}^+$ processes can be derived:

$$(\boldsymbol{\nu} a)(b/a \mid a/c) \;=\; b/c \qquad (\boldsymbol{\nu} a)(S + a(x).P) = (\boldsymbol{\nu} a)S$$
$$(\boldsymbol{\nu} a)(a/b \mid a(x).P) \;=\; \{b\}(x).(\boldsymbol{\nu} a)P \qquad \overline{a}(x).x = \overline{a}(x).\{x\} \qquad a(x).\{x\} = a(x).0$$

The first equality above is established using law L23: before getting rid of the restriction on a, we compute all conditions not involving a that can be deduced from $b/a \mid a/c$. In this case, this is only b/c.

The second equality is a direct consequence of law L25.

Law L25 is also used for the third equality: only the second summand in the law is not empty, which gives $(\boldsymbol{\nu} a)(a/b, a(x).P) = (\boldsymbol{\nu} a)(a/b, \{b\}(x).(\boldsymbol{\nu} a)P)$. Then, L20 allows us to restrict the scope of $\boldsymbol{\nu} a$, and we can get rid of $(\boldsymbol{\nu} a)a/b$ using law L23, which yields the result.

Another way to see the third equality is to observe that we can derive $a\!/\!b, a(x).P = a\!/\!b, a(x).P + \{a\}(x).P + \{b\}(x).P$ using laws L10 and L12. In the latter process, the sum is intuitively *expanded*, in the sense that all derivable toplevel summands have been made explicit. When considering the restricted version of both processes, it is sound to push the restriction on a downwards in the expanded process, to obtain the expected equality. In this sense, law L25 implements a "synchronous version" of this reasoning, so as to insure that when pushing a restriction downwards, the behaviour of the process is fully expanded.

The next two equalities illustrate the meaning of protected names. We reason as follows: $\overline{a}(x).x \overset{L18}{=} \overline{a}(x').(\boldsymbol{\nu}x)(x'\!/\!x, x) \overset{L25}{=} \overline{a}(x').(\boldsymbol{\nu}x)(x'\!/\!x, \{x'\}.(\boldsymbol{\nu}x)0)$. We then obtain the expected equality by getting rid of $(\boldsymbol{\nu}x)x'\!/\!x$, using laws L20-L23. The reason why this equality holds is that fresh name x is emitted without the context having the ability to interact at x, since x will never be under another name in an arc. Therefore, the input at x is equivalent to a protected input.

In the last equality, because of the transition $a(x).\{x\} \xrightarrow{a(x')} (\boldsymbol{\nu}x)(x'\!/\!x \mid \{x\})$, x will never be above another name, so that the prefix $\{x\}$ cannot be triggered, and is equivalent to 0. This equality is derived as follows:

$$a(x).\{x\} \overset{L17}{=} a(x').(\boldsymbol{\nu}x)(x'\!/\!x \mid \{x\}) \overset{L26}{=} a(x').(\boldsymbol{\nu}x)x'\!/\!x \overset{L23}{=} a(x').0 \ .$$

(we have explained above how $a(x').0 = a(x).0$ can be derived).

We leave it to the reader to check that the law for interleaving, presented in Section 4.4, can be derived using the expansion law, followed by the rules for prefixes and restriction to get rid of the summand $[x \curlyvee y]\tau.(\boldsymbol{\nu}t, u)(t\!/\!u)$.

4.6.2 Soundness and completeness of the axioms

Lemma 4.6.1 (Soundness)**.** *The laws of Table 4.2 relate bisimilar processes.*

Proof. Laws L19-L22 are instances of Lemma 4.5.26.

Laws L1-L4 are proven by showing $\Phi(lhs) = \Phi(rhs)$, since both sides have no transitions, which is done by a simple induction.

Similarly for law L23: the inclusion $\Phi(lhs) \supseteq \Phi(rhs)$ is trivial, and the inclusion $\Phi(lhs) \subseteq \Phi(rhs)$ is shown by induction, proving that all useful usages of a have to be combined to other usages of a using a technique similar as in case 3 of the proof of Lemma 4.5.15.

The remaining laws are easy: the clause about the preorder relation is always trivial, so we only focus on the clauses about transitions. In each case, we need only to run one step of the bisimulation game to reach processes we already know how to prove bisimilar.

For L8-L16 we check that lhs can do the transitions corresponding to the additional summands of rhs. Law L4 is trivial. For L5-L6 we need $\varphi \mid \varphi \sim \varphi$ which is derivable from L1-L4. For L17-L18 we get two pairs of the form $(\boldsymbol{\nu}y)(z\!/\!y \mid P)$ and $(\boldsymbol{\nu}x)(z\!/\!x \mid (\boldsymbol{\nu}y)(x\!/\!y \mid P))$ which is derivable using L23.

For L24-L29 we check that the transitions of both sides are the same. Then, we reach processes of the form $(\boldsymbol{\nu}a)(A, R)$ and $(\boldsymbol{\nu}a)(A \mid (\boldsymbol{\nu}a)(A \mid R))$. The latter process is bisimilar to $(\boldsymbol{\nu}a')(A\{a'\!/\!a\} \mid (\boldsymbol{\nu}a)(A \mid R))$, which is bisimilar to $(\boldsymbol{\nu}a)(A \mid ((\boldsymbol{\nu}a')(A\{a'\!/\!a\}) \mid R)$ again bisimilar to $(\boldsymbol{\nu}a)(A \mid ((\boldsymbol{\nu}a)A) \mid R)$ and we conclude by observing that $A \mid (\boldsymbol{\nu}a)A \sim A$ using the fact that $\Phi(A \mid (\boldsymbol{\nu}a)A) = \Phi(A)$, since $\Phi((\boldsymbol{\nu}a)A) \subseteq \Phi(A)$. □

Auxiliary results: preorder processes, prefixes, restriction.

In order to establish completeness, we first need some technical results, given by Propositions 4.6.2, 4.6.5 and 4.6.8.

First, laws L1-L4 can be used to *saturate* preorder processes:

Proposition 4.6.2. *If $A_1, S_1 \sim A_2, S_2$, then there exists $A^\star$ such that $\vdash A_i, S_i = A^\star, S_i$ $(i = 1, 2)$, and $A^\star = \prod\{\varphi \mid \varphi$ not reflexive and $A_1 \rhd \varphi\}$.*

(Note that we could have picked A_2 instead of A_1 above, and that $\prod\{P_1, \ldots, P_n\}$ stands for the parallel composition $P_1 \mid \ldots \mid P_n$.)

Proof. We rely on a rewriting relation on preorder processes. We write $A \overset{\Upsilon}{\mapsto} A'$ whenever A' is obtained from A using one of the laws L1-L4, oriented from left to right, as a rewrite rule modulo associativity and commutativity of parallel composition. We furthermore impose that no reflexive condition is added in a rewrite step, nor a condition that is already contained in the preorder process.

We then prove the following three properties about $\overset{\Upsilon}{\mapsto}$:

1. If $A \overset{\Upsilon}{\mapsto} A'$, then $A \sim A'$: this is a consequence of Lemma 4.6.1 (or, alternatively, follows from Lemma 4.5.29). This is useful to be able to relate $\overset{\Upsilon}{\mapsto}$-normal forms through $\sim$ and hence say that they entail the same conditions.

2. For any (finite) A, there is no infinite $\overset{\Upsilon}{\mapsto}$-chain emanating from A.

   Indeed, the rules defining $\overset{\Upsilon}{\mapsto}$ do not introduce any new name. Moreover, a new arc or join can only be added if it is not already present. Since there are finitely many conditions built on a finite set of names, $\overset{\Upsilon}{\mapsto}$ terminates.

3. Suppose A is a $\overset{\Upsilon}{\mapsto}$-normal form. Then, for any non-reflexive φ, if $A \rhd \varphi$, then φ appears in A (which we write $\varphi \in A$).

   This follows by induction on the derivation of $A \rhd \varphi$. The only interesting case is for the $\rhd$-COMBINE rule, i.e. we know that $A \rhd \Gamma$ and $\Gamma \vdash \varphi$. We conclude by associating to rules $\vdash$-TRANS, $\vdash$-JOIN and $\vdash$-EXTJOIN laws L1, L2 and L3 respectively.

The observations above entail the expected property.

□

We say that A is a *saturated preorder process* whenever $A^\star \equiv A$. We use $A^\star$ to range over such processes. We can remark that even if A contains only arcs, $A^\star$ may contain restrictions, because of induced conditions involving Υ.

The next lemma relates transitions of sums and the laws for prefixes.

Lemma 4.6.3. *If $A, S \xrightarrow{\mu} A, P$ then $\vdash A, S = A, S + \pi.Q$ for some π and Q such that μ and π only differ in their bound names and $\pi.Q \xrightarrow{\mu} P$.*

Proof. Suppose S is of the form $S_1 + \pi'.Q$ and that the transition $A, S \xrightarrow{\mu} A, P$ is in fact (Lemma 4.5.23) coming from the summand $\pi'.Q \xrightarrow{\mu'} P$, with $\mu \sqsubseteq_{A,S} \mu'$ and $\mu' =_\alpha \pi'$ (i.e. μ' and π' only differ in their bound names). Since $\Phi(A, S) = \Phi(A)$ we know also that $\mu \sqsubseteq_A \mu'$.

Then we have directly $\pi \sqsubseteq_A \pi'$. We prove by induction on $\pi \sqsubseteq_A \pi'$ that for all S, $\vdash A, S + \pi'.Q = A, S + \pi'.Q + \pi.Q$.

- Reflexivity of $\sqsubseteq$ is handled by the fact that $+$ is idempotent.

- Transitivity (e.g. $\pi_3 \sqsubseteq_A \pi_2 \sqsubseteq_A \pi_1$) is handled by monoid laws for $+$. We write Q_i for $\pi_i.Q$ below. We know by induction that:
  (1) $\vdash A, S + Q_1 = A, S + Q_1 + Q_2$ (for all S) and
  (2) $\vdash A, S + Q_2 = A, S + Q_2 + Q_3$ (for all S). Then (up to $\equiv$):

  $$\begin{aligned} \vdash\ & A, S + Q_1 \overset{(1)}{=} A, S + Q_1 + Q_2 \overset{(2)}{=} \\ & A, (S + Q_1) + Q_2 + Q_3 \overset{(1)}{=} A, S + Q_1 + Q_3 \end{aligned}$$

  so now we can only have to prove it for $\pi \sqsubseteq^\varphi \pi'$ when $A \rhd \varphi$. Note that using the reasoning above we can work up to transitivity.

- We now decompose $\sqsubseteq^{\varphi}$ when $A \rhd \varphi$. We know by Lemma 4.5.20 that there are some $\varphi_1, \ldots, \varphi_n \in A$ and a reflexive ψ such that $\sqsubseteq^{\varphi} = \sqsubseteq^{\psi} \sqsubseteq^{\varphi_1} \ldots \sqsubseteq^{\varphi_n}$ so in fact we only need to prove the result when φ is actually in A or when it is reflexive:
  - φ is reflexive and $\alpha(x) \sqsubseteq^{a \prec a} \alpha(x)$: nothing to do (idempotence of $+$)
  - φ is reflexive and $\{a\}(x) \sqsubseteq^{a \curlyvee a} a(x)$: law L10
  - $\varphi \in A$ and $\alpha(x) \sqsubseteq^{\beta \prec \alpha} \beta(x)$: this yields several cases:
    - $a(x) \sqsubseteq^{b \prec a} b(x)$: law L11
    - $\{a\}(x) \sqsubseteq^{a \prec b} \{b\}(x)$: law L12
    - $\{a\}(x) \sqsubseteq^{a \curlyvee b} b(x)$: we decompose $a \curlyvee b$ back into $u/a \mid u/b$ (law L23) then from $b(x)$ we get $u(x)$ by L11, then $\{u\}(x)$ (L10) and then $\{a\}(x)$ (L12).
  - φ is reflexive and $[\varphi_1]\tau \sqsubseteq^{\varphi} [\varphi_2]\tau$: we exploit L8 or idempotence of $+$.
  - $\varphi \in A$ and $[\varphi_1]\tau \sqsubseteq^{\varphi} [\varphi_2]\tau$ when $\varphi_1, \varphi \vdash \varphi_2$: this yields several cases again, we can decompose $\vdash$ into usages of $\Vdash$, reasoning up to transitivity. (We use instances of laws L16, L9, L15, L13.)

□

Laws L8-L16 can be used to "saturate" the topmost prefixes in sums. We express this using the equivalence below, and rely on Lemma 4.6.3 to prove Prop. 4.6.5:

Definition 4.6.4 (Head sum normal form, $\asymp_h$)**.** Given two sum processes S and T, we write $S \prec_h T$ whenever for any summand $\pi.P$ of S, there exists a summand $\pi.Q$ of T with $\pi.P \sim \pi.Q$. We let $S \asymp_h T$ stand for $S \prec_h T \wedge T \prec_h S$.

Proposition 4.6.5. *Whenever $A^\star, S_1 \sim A^\star, S_2$, where S_1, S_2 are two sum processes, there are S_1', S_2' s.t. $\vdash A^\star, S_i = A^\star, S_i'$ (for $i = 1, 2$) and $S_1' \asymp_h S_2'$.*

Proof. We first use law L5 to replicate $A^\star$ under all prefixes in S_1 and S_2, which is useful later in the proof. We therefore suppose that for any summand $\pi.P$ of S_1 or S_2, $P = A^\star \mid P_0$ for some P_0.

We prove the following property:

$$\begin{array}{l} A^\star, S_1 \sim A^\star, S_2 \\ \pi.P \in S_1 \end{array} \quad \Rightarrow \quad \exists Q \quad \begin{array}{l} \vdash A^\star, S_2 = A^\star, S_2 + \pi.Q \\ \pi.P \sim \pi.Q \end{array} \tag{4.5}$$

by running the bisimulation game with a label μ such that π and μ differ only on their object (that should be fresh in μ): $\pi.P \xrightarrow{\mu} P'$, yielding $A^\star, S_1 \xrightarrow{\mu} A^\star \mid P'$; the game gives a transition $A^\star, S_2 \xrightarrow{\mu} A^\star \mid Q'$. Using Lemma 4.6.3 we get Q such that $A^\star, S_2 = A^\star, S_2 + \pi.Q$ and $\pi.Q \xrightarrow{\mu} Q'$. We now have to prove that $\pi.P \sim \pi.Q$. There are two cases:

1. if μ is a visible action, then, by definition of $\sim$, we have $A^\star \mid P' \sim A^\star \mid Q'$. We can now observe that $P' \sim A^\star \mid P'$ and $Q' \sim A^\star \mid Q'$, because $A^\star$ has been replicated under prefixes. We thus deduce $P' \sim Q'$, which, by congruence, gives $\mu.P' \sim \mu.Q'$. Using the appropriate law among L17-18, we deduce $\pi.P \sim \mu.P'$ and $\pi.Q \sim \mu.Q'$, which implies $\pi.P \sim \pi.Q$.
2. if $\mu = [\varphi]\tau$ then $\pi = \mu$ and $P' = P$, $Q' = Q$. The bisimulation game yields $\varphi \mid A^\star \mid P' \sim \varphi \mid A^\star \mid Q'$ and by the same reasoning as before, $\varphi \mid P' \sim \varphi \mid Q'$. By congruence $[\varphi]\tau.(\varphi \mid P') \sim [\varphi]\tau.(\varphi \mid Q')$, and by L6, $[\varphi]\tau.P' \sim [\varphi]\tau.Q'$ i.e. $\pi.P \sim \pi.Q$.

We have now (4.5). Equation (4.5) implies that $\vdash A^\star, S_2 = A^\star, S_2 + T_2$ with $S_1 \prec_h S_2 + T_2$ and $T_2 \prec_h S_1$.

By reasoning symmetrically about $S_2 + T_2$, we obtain T_1 such that $\vdash A^\star, S_1 = A^\star, S_1 + T_1$ with $S_2 + T_2 \prec_h S_1 + T_1$ and $T_1 \prec_h S_2 + T_2$. Since we also have $S_1 \prec_h S_2 + T_2$, we can conclude $S_1 + T_1 \prec_h S_2 + T_2$ and thus $S_1' \asymp_h S_2'$ with $\vdash S_i' = S_i + T_i$.

□

Remark 4.6.6 (On the definition of $\asymp_h$)**.** In the definition of $\prec_h$, we impose $\pi.P \sim \pi.Q$, and not simply $P \sim Q$. The equivalence induced by the choice of the latter condition would indeed be too discriminating. To see why, consider $Q_1 = a(x).c/x$ and $Q_2 = a(x).0$. Obviously, $c/x \not\sim 0$. On the other hand, we have $Q_1 \sim Q_2$: after a $\xrightarrow{a(y)}$ transition on both sides, we must compare $(\boldsymbol{\nu} x)(c/x \mid y/x)$ and $(\boldsymbol{\nu} x)(y/x)$, and both are bisimilar to 0. In order to derive $\vdash Q_1 = Q_2$, we rely on the following property, which explains the shape of laws L17, L18: $a(y).P \sim a(y).Q$ iff $(\boldsymbol{\nu} y)(x/y \mid P) \sim (\boldsymbol{\nu} y)(x/y \mid Q)$.

Proposition 4.6.8 expresses that restrictions can be pushed inwards in processes. It refers to the following notion of measure on processes (which is useful to reason by induction on processes in the completeness proof):

Definition 4.6.7 (Measure on processes)**.** Given a $\pi\mathrm{P}^+$ process P, we define $|P|$ as the maximum number of prefixes in summands of P, i.e., $|\sum_i \pi_i.P_i| = \max_i (1 + |P_i|)$ (hence $|0| = 0$), $|(\boldsymbol{\nu} a)P| = |P|$, $|P \mid Q| = |P| + |Q|$, and $|a/b| = 0$.

Proposition 4.6.8. *For any A, S, a, there exist A' and S' such that $\vdash (\boldsymbol{\nu} a)(A, S) = A', S'$ and $|(\boldsymbol{\nu} a)(A, S)| \geq |A', S'|$.*

Proof. Using name extrusion, we pull all toplevel restrictions of A, S in order to derive $\vdash A, S = (\boldsymbol{\nu} \widetilde{a})(A_0, S_0)$, for some A_0, S_0 without toplevel restriction.

We then reason by induction over the number of names in $\widetilde{a}$. We apply laws L25-L24 from left to right, until name a does not appear free in any topmost prefix of the sum. At that point, since the restriction on a has been pushed under prefixes, a has no free occurrence in the sum. We can thus use name intrusion, so that law L23 can be applied to get rid of the restriction on a on the preorder part of the process.

This operation is iterated until restrictions are pushed under all prefixes, and law L22 can be used to get rid of the restriction.

□

Establishing completeness. The grammar $P ::= A, \sum_i \pi_i.P_i \;\mid\; (\boldsymbol{\nu} a)P$ defines what we call *$|$-free processes*: only arcs are composed, and the non-preorder part of processes is a sum.

Proposition 4.6.9 (Characterisation, without parallel composition)**.**
For all $|$-free processes P and Q, $P \sim Q$ iff $\vdash P = Q$.

Proof. The 'if' part follows from Lemma 4.6.1 and congruence of $\sim$.

Suppose now $P \sim Q$. We reason by induction on $|P| + |Q|$. By Proposition 4.6.8, there are sum-only processes P_0, Q_0 with no toplevel restriction such that $\vdash P = P_0$ and $\vdash Q = Q_0$.

We then reason up to associativity and commutativity of parallel composition to write $\vdash P_0 = A_1, S_1$ and $\vdash Q_0 = A_2, S_2$. We have $A_1, S_1 \sim A_2, S_2$, which gives, by Proposition 4.6.2, $\vdash A_i, S_i = A^\star, S_i$ for $i = 1, 2$, for some $A^\star$.

We can then apply Proposition 4.6.5 to deduce $\vdash A^\star, S_i = A^\star, S_i'$, for $i = 1, 2$, for some S_1', S_2' s.t. $S_1' \asymp_h S_2'$.

To sum up, we have proved until now $\vdash P = A^\star, S_1'$, $\vdash Q = A^\star, S_2'$, and $S_1' \asymp_h S_2'$.

We now prove, by induction over the number of summands in S_1', that for any such summand $\pi.T_1$, there is a summand $\pi.T_2$ in S_2' s.t. $\vdash \pi.T_1 = \pi.T_2$. Once this will be proved, we shall establish the same way the symmetrical property, which will allow us to deduce $\vdash S_1' = S_2'$.

Suppose then $\pi.T_1$ is a summand of S_1'.

We reason by case analysis on the shape of π, and suppose $\pi = a(x)$. We know, since $S_1' \asymp_h S_2'$, that there is a summand $a(x).T_2$ of S_2' such that $a(x).T_1 \sim a(x).T_2$. We have $\vdash a(x).T_i = a(y).(\boldsymbol{\nu} x)(y/x \mid T_i)$, for $i = 1, 2$, by law L17, induction hypothesis (on $(\boldsymbol{\nu} x)(y/x \mid T_1) \sim (\boldsymbol{\nu} x)(y/x \mid T_2)$) and congruence using the context $a(y).[\cdot]$.

Moreover, since $a(x).T_1 \sim a(x).T_2$, we know, by definition of bisimilarity with the transition labelled by $a(y)$, that $(\boldsymbol{\nu} x)(y/x \mid T_1) \sim (\boldsymbol{\nu} x)(y/x \mid T_2)$. This allows us to rely on the induction hypothesis to show

$\vdash (\boldsymbol{\nu} x)(y/x \mid T_1) = (\boldsymbol{\nu} x)(y/x \mid T_2)$ and hence $\vdash a(y).(\boldsymbol{\nu} x)(y/x \mid T_1) = a(y).(\boldsymbol{\nu} x)(y/x \mid T_2)$ which gives us, as announced, $\vdash \pi.T_1 = \pi.T_2$.

The other cases for the shape of π are treated similarly. □

We now move to the full calculus, by taking into account parallel composition. As is usually the case, this step relies on applications of the expansion law.

Lemma 4.6.10. *For any P, there exists a $|$-free process Q s.t. $\vdash P = Q$.*

Proof. First $\vdash P = A, P_1$ using the monoid laws for parallel composition. Then by induction on P_1 we build S such that $\vdash P_1 = S$ and $|P_1| = |S|$. There are only two cases: sums, and parallel compositions of two sums (by induction hypothesis), on which we apply the expansion law. Both preserve $|\cdot|$. □

We can then extend Proposition 4.6.9 to the whole $\pi\mathrm{P}^+$ calculus:

Theorem 4.6.11 (Axiomatisation of $\sim$)**.** *For all P and Q, $P \sim Q$ iff $\vdash P = Q$.*

Proof. The theorem follows from Proposition 4.6.9 and Lemma 4.6.10. □

Remark 4.6.12 (Normal forms)**.** The proofs in this section suggest that we can define a strategy to apply the laws of our axiomatisation, in order to rewrite a $\pi\mathrm{P}^+$ process P to its *normal form*, $\mathsf{nf}(P)$, so that $P \sim Q$ iff $\mathsf{nf}(P) = \mathsf{nf}(Q)$.

For preorder processes, the saturated form is a normal form for $\sim$: if $A_1 \sim A_2$, then, by Proposition 4.6.2, $\vdash A_1^\star = A_2^\star$. By contrast, the proof of Proposition 4.6.5 does not compute a canonical form for sum processes. For instance, from the equivalence

$$b/a \mid c/a \mid \overline{a}(x).0 \sim b/a \mid c/a \mid \overline{a}(x).0 + \overline{b}(x).0 \ ,$$

Proposition 4.6.5 rewrites these processes into $b/a \mid c/a \mid \overline{a}(x).0 + \overline{b}(x).0$, but not into $b/a \mid c/a \mid \overline{a}(x).0 + \overline{b}(x).0 + \overline{c}(x).0$, which could be seen as a normal form for $\sim$, obtained by saturating the sum. Actually, the normal form could even be

$$b/a \mid c/a \mid \overline{a}(x).0 + \overline{b}(x).0 + \overline{c}(x).0 + \overline{\{a\}}(x).0 + \overline{\{b\}}(x).0 + \overline{\{c\}}(x).0 \ ,$$

by virtue of several applications of (the counterpart for output of) law L10 with π_1 an output prefix and π_2 a protected output.

Related works The stateful nature of the preorder component of $\pi\mathrm{P}$ processes can be related to *frames* in the applied π-calculus [AF01] and Psi-calculi [BJPV09]. Arcs in $\pi\mathrm{P}$ can be seen in some sense as substitutions, but they differ from the active substitutions of applied π. The latter map variables to terms, while, in the tradition of fusion calculi, we only have (channel) names in $\pi\mathrm{P}$. Moreover, several arcs acting on the same name are allowed in $\pi\mathrm{P}$, while a substitution acts on at most one variable in applied π. For these reasons, the behavioural theories of $\pi\mathrm{P}$ and applied π are rather different. Liu and Lin's proof system for applied π [LL10] departs from our axiomatisation for $\pi\mathrm{P}$, but has in common the stateful component of processes.

4.6.3 Adapting our axiomatisation to explicit fusions

We can reuse the ideas presented above to describe an axiomatisation for barbed congruence in explicit fusions ([GW00, WG04] and Section 3.2.3). Accordingly, we add guarded sums and conditional synchronisation to the explicit fusion calculus along the lines of $\pi\mathrm{P}^+$. We also adopt a presentation of the calculus with primitive bound prefixes. The conditions and arcs of $\pi\mathrm{P}^+$ become explicit fusions.

$$\pi \ ::= \ \overline{a}(x) \ \mid \ a(x) \ \mid \ [a{=}b]\tau \qquad\qquad P \ ::= \ P \mid Q \ \mid \ (\boldsymbol{\nu} a)P \ \mid \ a{=}b \ \mid \ \sum_i \pi_i.P_i$$

Standard structural laws.

$$P \mid (Q \mid R) = (P \mid Q) \mid R \qquad P \mid Q = Q \mid P \qquad P \mid 0 = P$$

$$(\boldsymbol{\nu} a)(P \mid Q) = P \mid (\boldsymbol{\nu} a)Q \text{ and } (\boldsymbol{\nu} b)P = (\boldsymbol{\nu} a)P\{a/b\} \text{ when } a \notin \mathrm{fn}(P) \qquad (\boldsymbol{\nu} a)\sum_i \pi_i.P_i = \sum_{i \mid a \notin \mathrm{n}(\pi_i)} \pi_i.(\boldsymbol{\nu} a)P_i$$

Laws for fusions.

$$a{=}a = 0 \qquad a{=}b \mid a{=}c = a{=}b \mid a{=}c \mid b{=}c \qquad a{=}b = b{=}a \qquad (\boldsymbol{\nu} a)a{=}b = 0$$

Laws for prefixes.

$$a{=}b \mid S + \pi.P = a{=}b \mid S + \pi.(a{=}b \mid P) \qquad [a{=}b]\tau.P = [a{=}b]\tau.(a{=}b \mid P) \qquad \pi.P + \pi.P = \pi.P$$

$$[a{=}b]\tau.P = [b{=}a]\tau.P \qquad a{=}b \mid [a{=}c]\tau.P = a{=}b \mid [b{=}c]\tau.P \qquad a{=}b \mid S + a(x).P = a{=}b \mid S + b(x).P$$

$$a{=}b \mid S + \overline{a}(x).P = a{=}b \mid S + \overline{b}(x).P$$

Expansion law.

$$\begin{aligned} \textstyle\sum_i \pi_i.P_i \;\mid\; \sum_j \rho_j.R_j \quad &= \textstyle\sum_i \pi_i.(P_i \mid T) + \sum_j \rho_j.(S \mid R_j) \\ &\quad + \textstyle\sum_{i,j} [a{=}b]\tau.(\boldsymbol{\nu} x)(P_i \mid R_j) \\ &\quad \text{where } \{\pi_i, \rho_j\} = \{\overline{a}(x), b(x)\} \end{aligned}$$

Table 4.3: Axiomatisation for the explicit fusions calculus

As for $\pi\mathrm{P}^+$, free prefixes can be encoded: $[ab.P] = a(u).(u{=}b \mid P)$. Note in passing that fusions can be represented in $\pi\mathrm{P}^+$, encoding $a{=}b$ with $a\!/\!b \mid b\!/\!a$ (see Section 4.7.1).

We have defined in Section 3.2.3 an LTS and a definition of *efficient bisimulation* adapted[4] from [WG04]. The only difference with Section 3.2.3 are sums (for which we take the standard rule $\rightarrow$-SUM in Figure 4.1, page 59) and prefixes, for which we give the rules below:

$$\frac{}{\overline{a}(z).P \xrightarrow{\overline{a}(x)} (\boldsymbol{\nu} z)(z{=}x \mid P)} \qquad \frac{}{a(z).P \xrightarrow{a(x)} (\boldsymbol{\nu} z)(z{=}x \mid P)} \qquad \frac{}{[a{=}b]\tau.P \xrightarrow{[a=b]\tau} P}$$

We do not change the definition of bisimulation (cf. Remark 4.5.4 and Section 4.5.6).

We give in Table 4.3 an axiomatisation of efficient bisimulation (and hence, barbed congruence) for the explicit fusion calculus. We write $\vdash_{\mathrm{EF}} P = Q$ if the equality can be derived using equational reasoning from these laws.

The axiomatisation is considerably simpler than the one for $\pi\mathrm{P}^+$. The stateful component of processes encodes an equivalence relation on names. This makes it possible to reason locally, whereby a form of global reasoning is necessary in the laws of Table 4.2 for $\pi\mathrm{P}^+$. The fact that fusions are symmetric (and that the communicability relation is transitive) makes protected names unnecessary and simplifies greatly the handling of restriction. Indeed, laws L25-L29 can be dealt with using the last of the structural laws in Table 4.3.

The effect of an explicit fusion $a{=}b$ can be summed up with a simple derivable equality:

[4]Note that in the original LTS of [WG04], $P \xrightarrow{\tau} P'$ does not necessarily imply that $P \xrightarrow{[a=b]\tau} P'$ for every a and b, so the authors had to adapt the last clause of the definition of bisimulation (requiring $Q \mid a{=}b \xrightarrow{\tau}$ instead of $Q \xrightarrow{[a=b]\tau}$). Given the LTS of Section 3.2.3 we could have chosen the last clause of Definition 3.2.1 to be similar to the last clause of Definition 4.5.5.

Lemma 4.6.13. *For any P, a, b, we can derive $\vdash_{\mathrm{EF}} P \mid a{=}b = P\{a/b\} \mid a{=}b$.*

Proof. Every b in top-level object position can be rewritten into an a, using the last three prefix rules, and we propagate the fusion $a{=}b$ in depth using the first prefix law. □

Following the proofs of soundness and completeness for the axiomatisation of $\pi\mathrm{P}^+$, we derive the same result for explicit fusions:

Proposition 4.6.14. *For any P, Q, we have $P \sim_{\mathrm{ef}} Q$ iff $\vdash_{\mathrm{EF}} P = Q$.*

4.7 Expressiveness of $\pi\mathrm{P}$

We compare $\pi\mathrm{P}$ with other calculi, both as examples of the use of the calculus and as a test for its expressiveness. We study the explicit fusion calculus (Section 4.7.1) and two versions of the π-calculus (Section 4.7.2).

We will study $\pi\mathrm{P}$, and not $\pi\mathrm{P}^+$, as we do not need to assume the constructs of Sections 4.5 and 4.6, i.e. sums, conditional τs, extended names and bound prefixes. We revert back to the first syntax (Section 4.1) with the by-need semantics (Section 4.3).

When useful, we work in a *polyadic* version of $\pi\mathrm{P}$; the addition of polyadicity goes as for other name-passing calculi in the literature. Polyadic $\pi\mathrm{P}$ can be encoded into the monadic calculus along the lines of the analogous encoding for the π-calculus.

4.7.1 Explicit fusions

Bi-directional arcs, e.g., $a\!\!/b \mid b\!\!/a$, work as name fusions (cf, Lemma 4.3.14(3)). We thus can encode calculi based on name fusions into $\pi\mathrm{P}$. As an example, we consider the explicit fusion calculus presented in Section 3.2.3.

The encoding from explicit fusions to $\pi\mathrm{P}$ is defined as follows for prefixes and fusions, the other constructs being encoded homomorphically (names w and y are chosen fresh in the encodings of the output and, respectively, input prefixes below. We also adopt a polyadic notation for prefixes: $a\langle\tilde{v}\rangle$ and $\overline{a}\langle\tilde{v}\rangle.$):

$$\begin{array}{rcl} [\![a{=}b]\!] & = & a\!\!/b \mid b\!\!/a \\ [\![\overline{a}\langle v\rangle.P]\!] & = & (\boldsymbol{\nu} w)\overline{a}\langle v, w\rangle.w\langle v\rangle.[\![P]\!] \\ [\![a\langle x\rangle.Q]\!] & = & (\boldsymbol{\nu} y)a\langle x, y\rangle.\overline{y}\langle x\rangle.[\![Q]\!] \end{array}$$

In explicit fusions, a reduction step introduces a name fusion, say $a{=}b$. In the $\pi\mathrm{P}$ encoding, this is mimicked in two steps, so to be able to produce, correspondingly, two bidirectional arcs, $a\!\!/b$ and $b\!\!/a$. The first step installs the first arc. The second step is a communication on a private name, and has the effect of installing the reverse arc. We do not know whether the encoding is fully abstract. We present an operational correspondence result, given by Theorem 4.7.6 below.

In order to derive Theorem 4.7.6, we present a series of technical results. In the following, unless otherwise stated, P and Q range over $\pi\mathrm{P}$ processes, except when they appear inside encodings (e.g. in $[\![P]\!]$), in which case they range over processes of the explicit fusion calculus. We let $P \triangleright a = b$ stand for $P \triangleright a \prec b$ and $P \triangleright b \prec a$. We write $P =_{a,b} Q$ iff $P\{b/a\} = Q\{b/a\}$, i.e., P and Q only differ in some occurrences of names a and b. We recall that $\Phi(P)$ is the set $\{\varphi \mid P \triangleright \varphi\}$.

Lemma 4.7.1. *If $P =_{a,b} Q$, then $\Phi(P \mid [\![a{=}b]\!]) = \Phi(Q \mid [\![a{=}b]\!])$.*

Proof. First, as a consequence of the definition of the judgement $P \triangleright \varphi$, we can remark that Φ is monotonic (in the sense that $\Phi(P) \subseteq \Phi(P \mid Q)$), and that there is a function f such that $\Phi(P_1 \mid P_2) = f(\Phi(P_1), \Phi(P_2))$ (where f is monotonic). Let $A \triangleq [\![a{=}b]\!]$.

We prove the lemma by induction on P. The case where P does not contain any prefix is proved by establishing the simple laws $\Phi(a\!\!/c \mid A) = \Phi(b\!\!/c \mid A)$ and $\Phi(c\!\!/a \mid A) = \Phi(c\!\!/b \mid A)$. For example the first equality

can be obtained by remarking that $\Phi(a/c) = \Phi((\boldsymbol{\nu} b)(b/c \mid a/b)) \subseteq \Phi(b/c \mid a/b)$ (by Lemma 4.3.12, assuming $a \neq b$ and $b \neq c$ without loss of generality) and then that $\Phi(b/c \mid a/b \mid A) \subseteq \Phi(b/c \mid A \mid A) = \Phi(b/c \mid A)$ by monotonicity and idempotence (since $\Phi(P \mid P) = \Phi(P)$ for all P).

Using idempotence and Lemma 4.5.15, we deduce that $\Phi(P_1 \mid P_2 \mid A) = f(\Phi(P_1 \mid A), \Phi(P_2 \mid A))$. This observation allows us to treat all other cases of the induction. □

We extend the definition of $=_{a,b}$ to conditions: $\varphi =_{a,b} \psi$ iff $\varphi\{b/a\} = \psi\{b/a\}$. Lemma 4.7.1 can be slightly generalised:

Lemma 4.7.2. *If $P =_{a,b} Q$ and $\varphi =_{a,b} \psi$ then $P \mid [\![a{=}b]\!] \rhd \varphi$ iff $Q \mid [\![a{=}b]\!] \rhd \psi$.*

Proof. By Lemma 4.7.1 we only have to prove that if $R = S \mid [\![a{=}b]\!]$ then $R \rhd \varphi$ implies $R \rhd \psi$, which is easy, since for each case there is a rule from Definition 4.3.1 whereby a/b (resp. b/a) is used to replace an occurrence of a with b (resp. vice versa). □

Lemma 4.7.3. *If $P =_{a,b} Q$ then $(P \mid [\![a{=}b]\!]) \sim (Q \mid [\![a{=}b]\!])$.*

Proof. By an easy induction on $|P|$. The case for prefixes is important, but is easily handled using Lemma 4.5.23: since if $\mu_1 =_{a,b} \mu_2$ then $\mu_1 \sqsubseteq_{a=b} \mu_2$ (to prove the continuations bisimilar, we use the induction hypothesis).

When $P = P_1 \mid P_2$ (and then $Q = Q_1 \mid Q_2$ with $P_1 =_{a,b} Q_1$ and $P_2 =_{a,b} Q_2$) we just need to remark that $(P_1 \mid P_2) \mid [\![a{=}b]\!] \sim (P_1 \mid [\![a{=}b]\!]) \mid (P_2 \mid [\![a{=}b]\!])$ (and similarly for $Q_1 \mid Q_2$) to conclude by induction. □

Lemma 4.7.4. *If P and Q are prefix-free and $\Phi(P) = \Phi(Q)$ then $P \sim Q$.*

Proof. Straightforward, as P and Q have no transitions. □

Lemma 4.7.5. *For every process P of the explicit fusion calculus, if $[\![P]\!] \rhd a \prec b$ or $[\![P]\!] \rhd a \curlyvee b$, then $[\![P]\!] \rhd a = b$.*

Proof. First we prove that $[\![P]\!] \rhd a \prec b$ implies $[\![P]\!] \rhd b \prec a$ by induction on the derivation of the first judgement. The only interesting case is when we use an arc b/a: by definition of the encoding, we know that the arc a/b is present as soon as b/a is, which gives the expected property.

We then exploit this property in the case where $[\![P]\!] \rhd a \curlyvee b$. We know that there is a name u such that $a \prec u$ and $b \prec u$ and we use the first part of the proof to deduce $u \prec a$ and $u \prec b$, which yields $a \prec b$ and $b \prec a$. □

We recall that $\longrightarrow_{\mathrm{ef}}$ is the reduction relation in the explicit fusion calculus, defined in Section 3.2.3.

Theorem 4.7.6 (explicit fusions, operational correspondence)**.** *Let P, Q be processes of explicit fusions.*

1. *If $P \equiv Q$ then $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$;*
2. *if $P \longrightarrow_{\mathrm{ef}} P'$ then $[\![P]\!] \longrightarrow_{\mathrm{bn}} \cong_{\mathrm{bn}} [\![P']\!]$;*
3. *conversely, if $[\![P]\!] \longrightarrow_{\mathrm{bn}} Q$, then $Q \cong_{\mathrm{bn}} [\![P']\!]$ for some P' such that $P \longrightarrow_{\mathrm{ef}} P'$.*

A similar result holds for the fusion calculus [PV98a]. The present statement is simpler, because in explicit fusions the triggering of a reduction step does not depend on the presence of a restriction.

Proof. 1) We prove $[\![P]\!] \sim [\![Q]\!]$ by induction on the derivation of $P \equiv Q$. The standard base cases like associativity are treated easily, because the translation yields structurally congruent processes.

The other base cases are those dealing with fusions processes:

- law $[\![a{=}b \mid P]\!] \sim [\![a{=}b \mid P\{a/b\}]\!]$ follows by Lemma 4.7.3,
- laws $[\![a{=}a]\!] \sim [\![0]\!]$ and $[\![(\boldsymbol{\nu} a)a{=}b]\!] \sim [\![0]\!]$ follow by Lemma 4.7.4.

We conclude thanks to the fact that $\sim$ is a congruence and an equivalence relation.

2) Thanks to 1) and the fact that relation $\longrightarrow_{\mathrm{bn}}$ is preserved by evaluation contexts, we only have to consider the base case of the reduction relation: $R \triangleq \overline{a}b.P \mid ac.Q \longrightarrow_{\mathrm{ef}} b{=}c \mid P \mid Q \triangleq R'$.

Moreover, since $\approxeq_{\mathrm{bn}}$ is stable by $\equiv$ and evaluation contexts, we can restrict ourselves to considering the reduction $[\![R]\!] \longrightarrow_{\mathrm{bn}} (\boldsymbol{\nu} wy)(b{/}c \mid w{/}y \mid wb.[\![P]\!] \mid \overline{y}\langle c\rangle.[\![Q]\!])$. The latter process can only make a deterministic reduction to process $[\![R']\!] \mid (\boldsymbol{\nu} wy)(w{/}y)$, which is strongly bisimilar to $[\![R']\!]$ by Lemma 4.7.4.

3) The reduction $[\![P]\!] \longrightarrow_{\mathrm{bn}} P_1$ comes from a communication between two prefixes $[\![\overline{a}\langle v\rangle.Q]\!]$ and $[\![b\langle x\rangle.R]\!]$ where P must be of the form $P \equiv (\boldsymbol{\nu}\tilde{c})(S \mid \overline{a}\langle v\rangle.Q \mid b\langle x\rangle.R)$ and $[\![S]\!] \triangleright a \curlyvee b$. Lemma 4.7.5 gives $S \equiv S \mid a{=}b$ to get finally $P \equiv (\boldsymbol{\nu}\tilde{c})(S \mid \overline{a}\langle v\rangle.Q \mid a\langle x\rangle.R)$ and $P \longrightarrow_{\mathrm{ef}} (\boldsymbol{\nu}\tilde{c})(S \mid v{=}x \mid Q \mid R) \triangleq P'$. We relate $[\![P']\!]$ to P_1 through $\approxeq_{\mathrm{bn}}$ by the same reasoning as in 2. □

Remark 4.7.7 (The Solo Calculus)**.** The calculus of Solos [LV03] can be seen as the polyadic fusion calculus without continuations ([LV03] shows that Solos can encode continuations). We believe that using basically the same machinery, we can show that polyadic πP without continuations is able to encode continuations as well.

4.7.2 π-calculus

The embedding of the π-calculus into any existing fusion calculus is defined by translating the bound input construct as follows:

$$[\![a(x).P]\!] = (\boldsymbol{\nu} x)\, ax.[\![P]\!]$$

(the other constructs being translated homomorphically). The same encoding can be used for πP.

The encoding of π-calculus into fusions is not fully abstract for barbed congruence (Section 4.4). We do not know whether the encoding of the full π-calculus into πP is fully abstract. We present an operational correspondence result below (Section 4.7.2.1), and show in Section 4.7.2.2 that the encoding is fully abstract on $A\pi$, the asynchronous subset where no continuation is allowed after the output prefix.

4.7.2.1 Operational correspondence for synchronous processes

We use the following properties of the encoding:

Lemma 4.7.8 (Operational correspondence)**.** *Let P, Q be π-calculus processes.*

1. *$P \equiv Q$ iff $[\![P]\!] \equiv [\![Q]\!]$;*
2. *if $P \longrightarrow_\pi P'$ then $[\![P]\!] \longrightarrow_{\mathrm{bn}} \simeq_{\mathrm{bn}} [\![P']\!]$;*
3. *conversely, if $[\![P]\!] \longrightarrow_{\mathrm{bn}} P_1$ then there is P' such that $P \longrightarrow_\pi P'$ and $P_1 \simeq_{\mathrm{bn}} [\![P']\!]$;*
4. *$P \downarrow_a$ iff $[\![P]\!] \downarrow_a$.*

Proof. 1. The direction from left to right is straightforward. For the converse direction, suppose $[\![P]\!] \equiv R_1$, then there exists R such that $P \equiv R$ and we can obtain R_1 from $[\![R]\!]$ only by moving restrictions on input objects. Hence if $R_1 = [\![Q]\!]$, we have necessarily $R = Q$.

2. By induction on $P \longrightarrow_\pi P'$.

   Base case: if $\overline{a}b.P \mid a(x).Q \longrightarrow_\pi P \mid Q\{b/x\}$ we have $[\![\overline{a}b.P \mid a(x).Q]\!] \longrightarrow_{\mathrm{bn}} \boldsymbol{\nu} x([\![P]\!] \mid b{/}x \mid [\![Q]\!]) \simeq_{\mathrm{bn}} [\![P]\!] \mid [\![Q]\!]\{b/x\}$ by Lemma 4.3.14 since x has no negative occurrence in $[\![Q]\!]$.

   Inductive case: $E[P] \longrightarrow_\pi E[P']$ with $P \longrightarrow_\pi P'$ and E is either $R \mid [\cdot]$ or $(\boldsymbol{\nu} a)[\cdot]$. It is enough to observe that $[\![E]\!]$ is an evaluation context, since $\longrightarrow_{\mathrm{bn}}$ and $\simeq_{\mathrm{bn}}$ are both preserved by evaluation contexts.

   Inductive case: $P \equiv P_1 \longrightarrow_\pi P_1' \equiv P'$ with, by induction, $[\![P_1]\!] \longrightarrow_{\mathrm{bn}} \simeq_{\mathrm{bn}} [\![P_1']\!]$. Using the property established above, $[\![P]\!] \equiv \longrightarrow_{\mathrm{bn}} \simeq_{\mathrm{bn}} \equiv [\![P']\!]$. We conclude by remarking that $(\equiv \longrightarrow_{\mathrm{bn}} \simeq_{\mathrm{bn}} \equiv) \subseteq (\longrightarrow_{\mathrm{bn}} \simeq_{\mathrm{bn}})$, by definition of $\longrightarrow_{\mathrm{bn}}$ and $\simeq_{\mathrm{bn}}$.

3. Suppose $[\![P]\!] \longrightarrow_{\mathrm{bn}} P_1$. Since $[\![P]\!]$ does not contain any arc process, the reduction comes from a communication between two prefixes on the same name a: $[\![P]\!] \equiv E_1[\overline{a}b.[\![Q]\!] \mid ax.[\![R]\!]]$ with E binding x, and then, keeping track of all modifications brought by $\equiv$, we know that P_1 is of the form $P_1 \equiv E_1[[\![Q]\!] \mid b/x \mid [\![R]\!]]$. We can recover $P \equiv E[\overline{a}b.Q \mid a(x).R] \longrightarrow_\pi E[Q \mid R\{b/x\}] \triangleq P'$. Then $[\![P']\!] = [\![E]\!][[\![Q]\!] \mid [\![R]\!]\{b/x\}] \equiv E_1[[\![Q]\!] \mid [\![R]\!]\{b/x\}] \simeq_{\mathrm{bn}} P_1$.

   In the reasoning above, we rely on the following decomposition: if $[\![P]\!] \equiv E_1[\overline{a}b.Q_1 \mid ax.R_1]$ then $Q_1 \equiv [\![Q]\!]$, $R_1 \equiv [\![R]\!]$ and $P \equiv E[\overline{a}b.Q \mid a(x).R]$ with $[\![E]\!][\boldsymbol{\nu} x[\cdot]] \equiv E_1[\cdot]$. We prove this decomposition by combining techniques we have used in the first item, to deduce that Q_1 and R_1 are structurally congruent to the encodings of some processes, with the fact that structural congruence acts independently on either sides of the prefixes $\overline{a}b$ and ax.

4. The implication from left to right is straightforward by induction. Remark in passing that to test the input barb, a synchronous tester $\overline{a}b.\omega$ is needed (note that input barbs are not tested in the *asynchronous* version of behavioural equivalence.) The other implication follows from the fact that there is no arc in $[\![P]\!]$ so $[\![P]\!] \downarrow_a$ if and only if $[\![P]\!]$ contains a prefix whose subject is a (which is equivalent to P satisfying the same property).

□

One inclusion of the full abstraction result actually holds for the whole π-calculus:

Lemma 4.7.9. *Let P and Q be π terms. Then $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$ implies $P \simeq_\pi Q$.*

Proof. The relation $\{(P, Q) \mid [\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]\}$ is reduction-closed (consequence of Lemma 4.7.8), barb-preserving (consequence of Lemma 4.7.8), and context-closed: if C is a π context then there exists a $\pi\mathrm{P}$ context C_1 such that $[\![C[P]]\!] = C_1[[\![P]\!]]$, and similarly for Q. Hence $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$ implies $[\![C[P]]\!] \simeq_{\mathrm{bn}} [\![C[Q]]\!]$. □

4.7.2.2 Full abstraction for asynchronous processes

We now establish full abstraction for the encoding of $A\pi$, the asynchronous π-calculus [HT91, Bou92]. We say that $P \in \pi\mathrm{P}$ is *asynchronous* if the continuation of all outputs in P is 0 (more precisely, the only constraint we need is that no input is ever guarded by an output). We can remark that the encoding of a process in $A\pi$ is an asynchronous $\pi\mathrm{P}$ process.

We first establish correspondence results for labelled transitions. Labelled transitions in the π-calculus are written $\xrightarrow{\mu}_\pi$ (in particular, we write $\mu = a(x)$ for a *ground* input, i.e. when x is not free in the source process). For clarity, labelled transition in $\pi\mathrm{P}$ are written $\xrightarrow{\mu}_{\pi\mathrm{P}}$. The following lemma relates visible transitions in π to visible transitions in $\pi\mathrm{P}$ (for which we use the LTS of Section 4.5.6).

Lemma 4.7.10 (Label correspondences)**.** *Let P be any π process and f a fresh name.*

1. *If $P \xrightarrow{\overline{a}c}_\pi P'$ then $[\![P]\!] \xrightarrow{\overline{a}(y)}_{\pi\mathrm{P}} \sim c/y \mid [\![P']\!]$.*
2. *If $P \xrightarrow{\overline{a}(c)}_\pi P'$ then $[\![P]\!] \xrightarrow{\overline{a}(y)}_{\pi\mathrm{P}} \sim (\boldsymbol{\nu} c)(c/y \mid [\![P']\!])$.*
3. *If $P \xrightarrow{a(x)}_\pi P'$ then $[\![P]\!] \xrightarrow{a(y)}_{\pi\mathrm{P}} \sim (\boldsymbol{\nu} x)(y/x \mid [\![P']\!])$.*
4. *If $[\![P]\!] \xrightarrow{\overline{a}(y)}_{\pi\mathrm{P}} P_1$ then either $P \xrightarrow{\overline{a}c}_\pi P'$ with $P_1 \sim c/y \mid [\![P']\!]$, or $P \xrightarrow{\overline{a}(c)}_\pi P'$ with $P_1 \sim (\boldsymbol{\nu} c)(c/y \mid [\![P']\!])$.*
5. *If $[\![P]\!] \xrightarrow{a(y)}_{\pi\mathrm{P}} P_1$ then $P \xrightarrow{a(x)}_\pi P'$ with $P_1 \sim (\boldsymbol{\nu} x)(y/x \mid [\![P']\!])$.*

Moreover, protected labels are superfluous in the image of the encoding of π: if $[\![P]\!] \xrightarrow{\overline{\{a\}}(y)}_{\pi\mathrm{P}} P_1$ then $[\![P]\!] \xrightarrow{\overline{a}(y)}_{\pi\mathrm{P}} P_1$ and if $[\![P]\!] \xrightarrow{\{a\}(y)}_{\pi\mathrm{P}} P_1$ then $[\![P]\!] \xrightarrow{a(y)}_{\pi\mathrm{P}} P_1$.

Proof. Trivial, by transition induction. □

In general, when inspecting processes, it is always possible to decompose a τ transition into two visible transitions. In an asynchronous setting, we can moreover compose them back into a τ transition.

Lemma 4.7.11 (Re-composition of transitions, asynchronous πP)**.** *Let P be an asynchronous πP term. If $P \xrightarrow{\overline{\alpha}(x)}_{\pi\mathrm{P}} \xrightarrow{\beta(y)}_{\pi\mathrm{P}} P'$ then $P \xrightarrow{[\alpha \curlyvee \beta]\tau}_{\pi\mathrm{P}\sim} (\boldsymbol{\nu} xy)(P' \mid x/y)$.*

This result is the πP counterpart of the similar property in $A\pi$ (Lemma 5.3.2 in [SW01]). In the proof, we use $\sim$ for renaming and concatenating arcs involving fresh names using Lemma 4.3.12.

Lemma 4.7.12. *Let P and Q be $A\pi$ processes. Then $P \simeq_\pi Q$ implies $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$.*

Proof. We prove that $P \sim_{\mathrm{g}} Q$ implies $[\![P]\!] \sim [\![Q]\!]$, where $\sim_{\mathrm{g}}$ is ground bisimilarity in the π-calculus [SW01] (it is coarser than $\simeq_\pi$ in general, but coincides on $A\pi$ terms).

We do so by showing that the following relation is a bisimulation up to restriction and bisimilarity:

$$\mathcal{R} \triangleq \{([\![P]\!] \mid A, [\![Q]\!] \mid A) \;\mid\; P \sim_{\mathrm{g}} Q\} \ ,$$

where A is a parallel composition of arcs.

Clearly, $\Phi([\![P]\!] \mid A) = \Phi(A) = ([\![Q]\!] \mid A)$. Hence, we have to check the correspondences on transitions between π and πP, using Lemmas 4.7.8, 4.7.10 and 4.7.11. We analyse all possible transitions from $[\![P]\!] \mid A$:

1. in the case of an input transition, we have $[\![P]\!] \mid A \xrightarrow{\alpha(y)}_{\pi\mathrm{P}\sim} (\boldsymbol{\nu} x)(y/x \mid [\![P']\!]) \mid A$ with $P \xrightarrow{b(x)}_\pi P'$ such that $A \rhd \alpha \prec b$. We deduce from $P \sim_{\mathrm{g}} Q$ that $[\![Q]\!] \xrightarrow{b(y)}_{\pi\mathrm{P}\sim} (\boldsymbol{\nu} x)(y/x \mid [\![Q']\!])$ with $P' \sim_{\mathrm{g}} Q'$, so modulo $\sim$-rewriting and Lemma 4.5.23 we get a $\alpha(y)$ transition from $[\![Q]\!] \mid A$ and the resulting processes are related by $\sim\mathcal{R}^{\boldsymbol{\nu}}\sim$.

2. Symmetrically with an output transition, adding x/y instead of y/x to A.

3. The case of a silent transition $[\![P]\!] \mid A \xrightarrow{[\varphi]\tau}_{\pi\mathrm{P}} P_1 \mid A$ yields two cases:

   (a) either $[\![P]\!] \xrightarrow{\tau}_{\pi\mathrm{P}} P_1$. This is handled using Lemma 4.7.8.

   (b) or, in two steps (we write $\hat{c}$ for a set of names of size at most one, and $(\boldsymbol{\nu}\hat{c})P$ for $(\boldsymbol{\nu} c)P$ if $\hat{c} = \{c\}$ and P if $\hat{c} = \emptyset$ and similarly for label $(\boldsymbol{\nu}\hat{c})\overline{a}c$):

$$[\![P]\!] \xrightarrow{\overline{a}(y)}_{\pi\mathrm{P}} \xrightarrow{b(z)}_{\pi\mathrm{P}} (\boldsymbol{\nu}\hat{c}x)(c/y \mid z/x \mid [\![P']\!]) \triangleq P_2$$

$$P \xrightarrow{\boldsymbol{\nu}\hat{c}\overline{a}c}_\pi \xrightarrow{b(x)}_\pi P'$$

   such that $[\varphi]\tau \sqsubseteq_A [a \curlyvee b]\tau$ and $P_1 \sim (\boldsymbol{\nu} yz)(P_2 \mid y/z)$.

   We can again play the ground bisimilarity game, first with transition $\boldsymbol{\nu}\hat{c}\,\overline{a}c$ and then with transition $b(x)$ to get $Q \xrightarrow{\boldsymbol{\nu}\hat{c}\overline{a}c}_\pi \xrightarrow{b(x)}_\pi Q'$ with $P' \sim_{\mathrm{g}} Q'$. Using Lemma 4.7.10 we obtain two transitions along $\overline{a}(y)$ and $b(z)$ from $[\![Q]\!]$, which we compose with Lemma 4.7.11 to get $[\![Q]\!] \mid A \xrightarrow{[a \curlyvee b]\tau}_{\pi\mathrm{P}} (\boldsymbol{\nu} yz)(\boldsymbol{\nu}\hat{c}x)((z/x \mid c/y \mid [\![Q']\!]) \mid y/z)$. Bisimilarity can be used to finally rewrite processes related through $\mathcal{R}^{\boldsymbol{\nu}}$:

$$\boldsymbol{\nu} yz\hat{c}x([\![P']\!] \mid A') \;\; \mathcal{R}^{\boldsymbol{\nu}} \;\; \boldsymbol{\nu} yz\hat{c}x([\![Q']\!] \mid A')$$

   with $A' = A \mid c/y \mid y/z \mid z/x$. We get back the $[\varphi]\tau$ transition using Lemma 4.5.23 and use $\sim$-rewriting again to push the additional φ into A'.

Relation $\mathcal{R}$ is symmetric, and thus $\mathcal{R}$ is a bisimulation up to restriction and bisimilarity; we conclude by Lemma 4.5.32. □

Theorem 4.7.13 (Full abstraction for the asynchronous π-calculus)**.** *Suppose P, Q are $A\pi$ processes. Then $P \simeq_\pi Q$ iff $[\![P]\!] \simeq_{\mathrm{bn}} [\![Q]\!]$.*

Proof. Follows from Lemmas 4.7.9 and 4.7.12. □

In the theorem, $\simeq_\pi$ is barbed congruence in the full π-calculus, as opposed to *asynchronous* barbed congruence, where output barbs are observable but input barbs are not. It can be proved that in a π-calculus with neither replication nor choice, asynchronous and synchronous barbed congruence coincide, and hence the theorem holds also under asynchronous barbed congruence.

Theorem 4.7.13 (and the results on types) shows that $\pi\mathrm{P}$, although syntactically similar to fusions, is closer to the π-calculus in the management of names.

Discussion. We do not know whether full abstraction holds for the encoding of the full π-calculus. More precisely, if the source calculus features replication or the choice operator, full abstraction would break at least for ground, early and late bisimilarities. The situation is less clear for open bisimilarity.

Regarding translations in the opposite direction, both for fusion calculi and for $\pi\mathrm{P}$, the encoding into π is not possible in general. Under some conditions, an arc $a\!\!/b$ can be encoded as the π-calculus process $!b(x).\overline{a}x$, called a *forwarder* from b to a. For this encoding to work, two constraints on the source calculus are necessary: *asynchrony* (all outputs have 0 as continuation) and *locality* (a name that has negative occurrences may not occur as subject of an input; in the π-calculus, this may be rephrased by saying that a name bound by an input may not occur as subject of an input). M. Merro presents in [Mer00] an encoding of the asynchronous monadic fusion calculus into $A\pi$. Such an encoding for $\pi\mathrm{P}$ would probably bear some resemblance to an encoding of the calculus obeying constraints of asynchrony and locality into $\mathrm{AL}\pi$, the asynchronous localised π-calculus [Mer00].

4.8 Unique negative occurrences of names

In this section we consider a constrained version of the calculi discussed until now, where each name may have at most one negative occurrence in a process. In the fusion calculus [PV98a] the constraint means that each name appears at most once as the object of an input. In $\pi\mathrm{P}$, the constraint affects also arcs, as their source is a negative occurrence. We call $\pi\mathrm{P1}$ and FU1 the constrained versions of $\pi\mathrm{P}$ and fusions. (Note that in the case of a calculus with replications, if a free name of P appears negatively in P, then $!P$ does not qualify as a constrained process.)

It is also possible to adapt this constraint to explicit fusions. This requires, given a process P of explicit fusions, to assign a "polarisation", for each explicit fusion $a{=}b$ occurring in P, whereby either a or b is considered as negative. Then P belongs to the constrained explicit fusion calculus if every name occurs at most once in negative position, either as an input object or in the negative side of an explicit fusion construct. Structural congruence has to be slightly adapted, as the reflexivity law $a{=}a \equiv 0$ does not preserve the constraint. In fact, removing it is enough, since it has no effect on the induced equivalence, since symmetry and transitivity are still derivable without it, and since, interestingly, the other laws preserve the constraint, in particular $P \mid a{=}b \equiv P\{b/a\} \mid a{=}b$, even if it might change the polarisation of the fusion $a{=}b$. We call the resulting calculus EF1.

The constraint is rather draconian, bringing the calculi closer to the π-calculus, where the constraint is enforced by having binding input. It may be remarked that when encoding the π-calculus in $\pi\mathrm{P}$ (see Section 4.7.2), the image is included in $\pi\mathrm{P1}$. Still, the constraint is more generous than tying the input to a binder as in π. For instance, we have more complex forms of causality involving input, as in the process $\boldsymbol{\nu}x(ax.\overline{w}t \mid \overline{b}x)$, where the input at a blocks the output at w, and can be triggered before or after the output at b takes place. Delayed inputs [Mer00] are thus simply encodable in $\pi\mathrm{P1}$ as $(\boldsymbol{\nu}x)\,(ax \mid P)$, whereas in π they represent a non trivial extension of the calculus.

In the remainder of this section, we show the consequences of imposing unique negative occurrences of names for behavioural equivalence (Section 4.8.1) and types (Section 4.8.2).

4.8.1 Behavioural properties of constrained calculi

By Lemma 4.1.3, a πP1 process can only reduce to a πP1 process. This allows us to adapt (weak) behavioural equivalence ($\cong_{\mathrm{bn}}$, Definition 4.3.7) to πP1, yielding $\cong_{\pi\mathrm{P1bn}}$, where in the last clause, we only allow contexts which, when applied to the processes being compared, yield πP1 processes. Similarly, $\cong_{\mathrm{ea}}$, the eager equivalence (Definition 4.1.1), yields $\cong_{\pi\mathrm{P1ea}}$.

We show that the constraint makes certain differences between calculi or semantics disappear. First, in the eager semantics for πP1, reduction steps where arcs are used to rewrite prefixes are deterministic.

Lemma 4.8.1 (Eager reduction in πP1)**.** *Consider $P \in \pi\mathrm{P1}$, and write $P \stackrel{/}{\longrightarrow} P'$ whenever $P \longrightarrow_{\mathrm{ea}} P'$ where the reduction is a rewrite step involving an arc. Then $P \stackrel{/}{\longrightarrow} P'$ implies $P \cong_{\pi\mathrm{P1ea}} P'$.*

Proof. In πP1, every name has at most one father in the partial order, by definition of πP1, which has two important consequences. The first consequence is that P and P' have the same barbs when $P \stackrel{/}{\longrightarrow} P'$. The second is that $\stackrel{/}{\longrightarrow}$ satisfies the diamond property in πP1.

This allows us to show that $\mathcal{R} = \{(P, P').\, P \stackrel{/}{\longrightarrow} P'\} \cup \{(P, P)\} \subseteq \cong_{\pi\mathrm{P1ea}}$. The interesting cases arise from the reductions of P. If $P \stackrel{/}{\longrightarrow} P_1$, then the diamond property allows us to conclude. If $P \longrightarrow_{\mathrm{ea}} P_1$ through a communication, then either $P' \stackrel{/}{\longrightarrow} \longrightarrow_{\mathrm{ea}} P_1$ (in the case where the communication involves the prefix that has been rewritten using $\stackrel{/}{\longrightarrow}$), or $P' \longrightarrow_{\mathrm{ea}} P'_1$ with $P_1 \stackrel{/}{\longrightarrow} P'_1$. In both cases, we close the diagram. □

Our proof actually shows that *P expands P'* (expansion being the efficiency preorder of [AKH92], defined precisely in Definition 5.1.17). In addition to Lemma 4.8.1, we can also observe that in πP1, we have the law $(\boldsymbol{\nu} a)a/b \cong_{\pi\mathrm{P1ea}} 0$, which follows from Theorem 4.8.2 and Example 4.3.10.

In fact, since reduction steps involving rewritings in the eager semantics do not correspond to commitments, the two versions of weak equivalence in πP1 coincide:

Theorem 4.8.2 (Coincidence of eager and by-need semantics in πP1)**.** $\cong_{\pi\mathrm{P1ea}} = \cong_{\pi\mathrm{P1bn}}$.

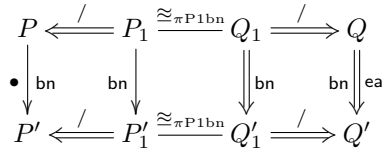
Proof. We first prove two commutation properties, where we write, like above, $P \stackrel{/}{\longrightarrow} P'$ when $P \longrightarrow_{\mathsf{ea}} P'$ follows from a rewriting step involving an arc, and $P \stackrel{\bullet}{\longrightarrow} P'$ when $P \longrightarrow_{\mathsf{ea}} P'$ follows from a communication step. We have the following:

1. $(\stackrel{/}{\longleftarrow}\longrightarrow_{\mathsf{bn}}) \subseteq (\longrightarrow_{\mathsf{bn}}\stackrel{/}{\longleftarrow}^{=})$ (which implies $(\stackrel{/}{\Longleftarrow}\Longrightarrow_{\mathsf{bn}}) \subseteq (\Longrightarrow_{\mathsf{bn}}\stackrel{/}{\Longleftarrow})$). (Here we write $\mathcal{R}^{=}$ for the extension of $\mathcal{R}$ with the reflexive relation.) The interesting case is when $\stackrel{/}{\longrightarrow}$ rewrites a name a to b, and a is involved in the communication taking place in the $\longrightarrow_{\mathsf{bn}}$ step (which exploits $a \curlyvee c$). In this case, the $\longrightarrow_{\mathsf{bn}}$ step can simulate the rewriting step (since $b \curlyvee c$), because $\stackrel{/}{\longrightarrow}$ makes no commitment.

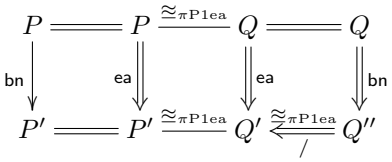
2. $(\stackrel{/}{\longrightarrow}\longrightarrow_{\mathsf{bn}}) \subseteq (\longrightarrow_{\mathsf{bn}}\stackrel{/}{\longrightarrow}^{=})$ (which implies $(\stackrel{/}{\Longrightarrow}\longrightarrow_{\mathsf{bn}}) \subseteq (\longrightarrow_{\mathsf{bn}}\stackrel{/}{\Longrightarrow})$). Using the same kind of reasoning, if $\stackrel{/}{\longrightarrow}$ rewrites a name a into b that is used afterwards in the communication step of $\longrightarrow_{\mathsf{bn}}$ (using $b \curlyvee c$), then $\longrightarrow_{\mathsf{bn}}$ can use the property $a \curlyvee c$.

To establish $\cong_{\pi\mathrm{P1bn}} \subseteq \cong_{\pi\mathrm{P1ea}}$, we prove that $\mathcal{R} = (\stackrel{/}{\Longleftarrow}\cong_{\pi\mathrm{P1bn}}\stackrel{/}{\Longrightarrow})$ satisfies the following stability property under eager reductions: $(\longleftarrow_{\mathsf{ea}}\mathcal{R}) \subseteq (\mathcal{R}\Longleftarrow_{\mathsf{ea}})$. Consider P, P_1, Q_1, Q and P' such that $P \longrightarrow_{\mathsf{ea}} P'$ and $P \stackrel{/}{\Longleftarrow} P_1 \cong_{\pi\mathrm{P1bn}} Q_1 \stackrel{/}{\Longrightarrow} Q$. We must show that we can find some Q' to close the diagram.

- The case $P \stackrel{/}{\longrightarrow} P'$ is straightforward (we choose $Q' = Q$).
- Suppose then $P \stackrel{\bullet}{\longrightarrow} P'$, which implies $P \longrightarrow_{\mathsf{bn}} P'$. By property 2 above, we get $P_1 \longrightarrow_{\mathsf{bn}} P'_1$ and $P'_1 \stackrel{/}{\Longrightarrow} P'$. This allows us to play the $\cong_{\pi\mathrm{P1bn}}$ game, and obtain $P'_1 \cong_{\pi\mathrm{P1bn}} Q'_1$ such that $Q_1 \Longrightarrow_{\mathsf{bn}} Q'_1$, which, thanks to property 1, gives $Q'_1 \stackrel{/}{\Longrightarrow} Q'$ with $Q \Longrightarrow_{\mathsf{bn}} Q'$ (hence $Q \Longrightarrow_{\mathsf{ea}} Q'$ by Lemma 4.3.4). The reasoning is depicted on the following diagram:



To prove $\approx_{\pi\mathrm{P1ea}} \subseteq \approx_{\pi\mathrm{P1bn}}$ we show that $\approx_{\pi\mathrm{P1ea}}$ enjoys the following property of stability under by-need reductions: $(\longleftarrow_{\mathsf{bn}} \approx_{\pi\mathrm{P1ea}}) \subseteq (\approx_{\pi\mathrm{P1ea}} \Longleftarrow_{\mathsf{bn}})$. Suppose that $P \approx_{\pi\mathrm{P1ea}} Q$ and $P \longrightarrow_{\mathsf{bn}} P'$ which implies by Lemma 4.3.4 that $P \Longrightarrow_{\mathsf{ea}} P'$. This allows us to draw the $\approx_{\pi\mathrm{P1ea}}$ diagram, to get $Q \Longrightarrow_{\mathsf{ea}} Q'$ such that $P' \approx_{\pi\mathrm{P1ea}} Q'$. Since $\longrightarrow_{\mathsf{ea}} = (\xrightarrow{\bullet} \cup \xrightarrow{/})$ and $\xrightarrow{\bullet} \subseteq \longrightarrow_{\mathsf{bn}}$, we deduce $\Longrightarrow_{\mathsf{ea}} \subseteq (\longrightarrow_{\mathsf{bn}} \cup \xrightarrow{/})^*$ and from property 2 above we have $\Longrightarrow_{\mathsf{ea}} \subseteq \Longrightarrow_{\mathsf{bn}} \xRightarrow{/}$, i.e., $Q \Longrightarrow_{\mathsf{bn}} Q'' \xRightarrow{/} Q'$ for some Q''. We are able to close the diagram with Q'' since $\xRightarrow{/} \subseteq \approx_{\pi\mathrm{P1ea}}$ by Lemma 4.8.1. Finally, since a name has a negative occurrence in Q' if and only if it has one in Q'', we have that $P' \approx_{\pi\mathrm{P1ea}} Q' \approx_{\pi\mathrm{P1ea}} Q''$ implies $P' \approx_{\pi\mathrm{P1ea}} Q''$.



We thus proved that the reduction diagrams for $\approx_{\pi\mathrm{P1ea}}$ and $\approx_{\pi\mathrm{P1bn}}$ can be related. Properties involving context closure and barbs correspond to each other in the eager and by-need cases, so that $\approx_{\pi\mathrm{P1ea}} = \approx_{\pi\mathrm{P1bn}}$. □

It also appears that the calculi πP1 and FU1, defined by the constraint on unique negative occurrence of names, are behaviourally close. For instance, in πP1 the direction of arcs is irrelevant (that is, arcs behave like explicit fusions), as shown by the following law (where we omit the subscripts 'ea' and 'bn' in the light of Theorem 4.8.2).

Lemma 4.8.3. *$a/b \approx_{\pi\mathrm{P1}} b/a$.*

Proof. The relation relating $C[\pi.E[a/b]\{a/x\}]$ with $C[\pi.E[b/a]\{b/x\}]$ (for any context C, any evaluation context E, and where x can only appear in subject position in E) is reduction-closed and barb preserving. □

Note that due to the constraint, the processes in Lemma 4.8.3 must be compared in contexts where a and b only occur positively.

We now prove that contextual equivalences in EF1 and πP1 coincide. For this, when $P \in \pi\mathrm{P1}$, we write $\overline{P} \in \mathrm{EF1}$ for the canonical projection of P in EF1. We moreover refine the behavioural equivalences as follows: we write $P \approx^N_{\pi\mathrm{P1}} Q$ iff $P \approx_{\pi\mathrm{P1}} Q$ and all names which occur free in negative position in P and Q belong to the set of names N. Moreover, when closing by contexts in checking $\approx^N_{\pi\mathrm{P1}}$, we impose that contexts cannot use names in N in negative position.

Relation $\cong^N_{\mathrm{EF1}}$ is defined along the same lines; for context closure, we impose that there exists a polarisation of each fusion construct such that the constraints about N and about uniqueness of negative occurrence are satisfied.

Lemma 4.8.4. *For all N, if all names occurring free in negative position in P and Q belong to N, we have $P \approx^N_{\pi\mathrm{P1}} Q$ iff $\overline{P} \cong^N_{\mathrm{EF1}} \overline{Q}$.*

For example Lemma 4.8.3 equivalently states that $a/b \approx^{\{a,b\}}_{\pi\mathrm{P1}} b/a$.

Proof. We prove the two implications:

- $\{(P,Q) \mid \overline{P} \cong^N_{\mathrm{EF1}} \overline{Q}\} \subseteq \approx^N_{\pi\mathrm{P1}}$ for all N:

- context closure: if C is a πP1 context without negative occurrence in N, there exists a EF1 context $\overline{C}$ such that $\overline{C}[\overline{R}] = \overline{C[R]}$ and there is a trivial polarisation of $\overline{C}$ such that no name having a negative occurrence in $\overline{C}$ is in N. Then if we write N_C for the names occurring free in negative position in C, we have that $\overline{C}[\overline{P}] \cong_{\text{EF1}}^{N \cup N_C} \overline{C}[\overline{Q}]$, which means $C[P]$ and $C[Q]$ are in the relation.
- reduction closure follows from the following operational correspondence:
  - if $P \longrightarrow_{\pi\text{P}} P'$ then $\overline{P} \rightarrow_{EF} \overline{P'}$ (indeed if $P \rhd a \curlyvee b$ then $\overline{P} \rhd a = b$)
  - if $\overline{P} \rightarrow_{EF} P_1$ then for some P', $P \rightarrow_{\pi P} P'$ and $P_1 = \overline{P'}$ (indeed if $\overline{P} \rhd a = b$ then $P \rhd a \curlyvee b$ since $P \in \pi\text{P1}$)
- the above two results allow us to deduce barb preservation.

- $\{(P_1, Q_1) \mid P_1 \stackrel{\leftrightarrow}{=} \overline{P} \wedge Q_1 \stackrel{\leftrightarrow}{=} \overline{Q} \wedge P \approx_{\pi\text{P1}}^{N} Q\} \subseteq \cong_{\text{EF1}}^{N}$ for all N, where $\stackrel{\leftrightarrow}{=}$ is the smallest congruence on EF1 terms satisfying $a{=}b \stackrel{\leftrightarrow}{=} b{=}a$ for all a, b.
  - context closure: if there some EF1 context C without negative occurrence in N, then for some πP1 context D we know that: for all $R \in \pi\text{P1}$ such that names that have a negative occurrence in R are in N, $\overline{D[\overline{R}]} \stackrel{\leftrightarrow}{=} C[\overline{R}]$. By context closure of $\approx_{\pi\text{P1}}^{N}$, we know that $D[P] \approx_{\pi\text{P1}}^{N \cup N_D} D[Q]$ and hence $(C[P_1], C[Q_1])$ is in the relation.
  - reduction closure and barb preservation are handled in the same way as above (note that $\stackrel{\leftrightarrow}{=}$ is a strong bisimulation).

□

4.8.2 Capabilities and subtypes in constrained fusions

Another difference that disappears under the constraint of unique negative occurrences of names is the one regarding capabilities and subtyping. We have seen that fusion calculi cannot have the same type systems as π and πP have (Sections 3.4 and 4.2). Indeed, it is possible to equip FU1 with an I/O type system and subtyping. To achieve this, we can use exactly the rules of πP in Section 4.2—with the exception of T-ARC as FU1 does not have arcs.

To understand how I/O types work in FU1, let us analyse how the constraint on unique occurrences of negative names evolves along reductions. We first consider the following reduction step in FU1 ($\longrightarrow_{\text{FU1}}$ stands for reduction in FU1):

$$(\boldsymbol{\nu} c)(\overline{a}b.P \mid ac.Q \mid R) \longrightarrow_{\text{FU1}} (P \mid Q \mid R)\{b/c\} \ . \tag{4.6}$$

In this reduction, only positive occurrences of c are replaced with b, since the only negative occurrence of c is in ac in the initial process. Hence, no negative occurrence of b is introduced.

The other kind of reduction in FU1 is as follows (note that the restriction is now on b, and that the substitution intuitively goes in the opposite direction w.r.t. πP):

$$(\boldsymbol{\nu} b)(\overline{a}b.P \mid ac.Q \mid R) \longrightarrow_{\text{FU1}} (P \mid Q \mid R)\{c/b\} \ . \tag{4.7}$$

Along reduction, the negative occurrence of c in prefix ac disappears. In the original process, the only visible usage of b is in $\overline{a}b$, which is a positive occurrence. Therefore, there is at most one negative occurrence of b in P, Q, R, hence $(P \mid Q \mid R)\{c/b\}$ contains at most one negative occurrence of c. The resulting process is hence a FU1 process. Finally, note that the following reduction in FU1 (the special case, when $b = c$) only makes prefixes disappear, and straightforwardly yields a process in FU1:

$$\overline{a}b.P \mid ab.Q \longrightarrow_{\text{FU1}} P \mid Q \ .$$

Polarised narrowing (Theorem 4.2.1) holds in FU1 because every process of FU1 is also a process of πP. Regarding the Subject Reduction property, we observe that typechecking the process obtained after reduction in (4.7) may involve changing the type of name c into a smaller type: after the reduction, name c is used at type T_b, which is a smaller type than the type of c. Accordingly, the statement of Subject Reduction is refined:

Theorem 4.8.5. *Let P be a* FU1 *process. If $\Gamma \vdash P$ and $P \longrightarrow_{\text{FU1}} P'$, then $\Gamma' \vdash P'$, where for at most one name c, $\Gamma'(c) \leq \Gamma(c)$; for any other name $b \neq c$, $\Gamma'(b) = \Gamma(b)$.*

Proof. By induction on $\longrightarrow_{\text{FU1}}$, only the base case is interesting (closure by contexts and structural congruence are handled by compositionality of the type system). There are two subcases, depending on whether the object being replaced is an input object or an output object.

Suppose we have the following typings in each case, which we write with additional type annotations:

1. $\Gamma, b : T_b \vdash (\boldsymbol{\nu} c : T_c)(\overline{a}b.P \mid ac.Q \mid R)$ (using (4.6) the process reduces to $(P \mid Q \mid R)\{b/c\}$),
2. $\Gamma, c : T_c \vdash (\boldsymbol{\nu} b : T_b)(\overline{a}b.P \mid ac.Q \mid R)$ (using (4.7) the process reduces to $(P \mid Q \mid R)\{c/b\}$).

By definition of FU1, P, Q, and R have no negative occurrence of c. Moreover, because of the typing rules, we have $T_b \leq T_c$.

In the first case, since c does not appear in negative position in $P \mid Q \mid R$, applying positive narrowing on $T_b \leq T_c$ is enough to conclude without changing the typing environment.

In the second case, we replace b with c, and assign type T_b to c in the typing environment for the resulting process. This allows us to typecheck the newly created occurrences of c in the resulting process. The previously existing ones are still well-typed using positive narrowing. □

Remark 4.8.6. Theorem 4.8.5 does not contradict Theorems 3.4.3 and 3.4.4: the calculus obeys a syntactical constraint, which restricts the way processes can be composed. Indeed, it is not always the case that if P and Q are in FU1, then $P \mid Q$ is also in FU1. Therefore, in the proof of Theorem 3.4.3, it would be impossible in general to replace a with b using contexts like $(\overline{u}b \mid ua \mid -)$ or $(\overline{u}b \mid \overline{v}a \mid uc \mid vc \mid -)$, since the latter is not a FU1 context.

4.9 Conclusion

This chapter is a study of πP, covering types, strong behavioural theory, a proof system, and the relationship to other calculi and variants of πP. Several questions remain, and we discuss some of them in this section.

Typed behavioural theory The results of Section 4.5 establish foundations for the behavioural theory of the πP calculus, and Section 4.2 studies i/o-types. These provide a basis to build a notion of typed behavioural equivalence [PS96, DS06], which can be used to establish fine behavioural properties of concurrent systems, like in Section 3.3.3. This could reveal helpful to refine untyped analyses of systems where fusions have been used.

Weak behavioural theory The first step in studying weak behavioural theory of πP is to build a the weak variant of bisimilarity in order to capture $\cong_{\text{bn}}$ with a characterisation theorem similar to Theorem 4.5.38. We define the candidate relation $\approx$ below and we comment on it:

Definition 4.9.1. A symmetric relation $\mathcal{R}$ is a weak bisimulation if $P \mathcal{R} Q$ implies:

1. If $P \rhd \varphi$ then $Q \Longrightarrow Q'$ and $Q' \rhd \varphi$ for some Q' such that $P \mathcal{R} Q'$.
2. If $P \xrightarrow{\alpha(x)} P'$, with $x \notin \text{fn}(Q)$, then there is Q' such that $Q \overset{\alpha(x)}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$; we impose the same condition with $\overline{\alpha}$ instead of α.
3. If $P \xrightarrow{[\varphi]\tau} P'$ then there is Q' such that $Q \mid \varphi \Longrightarrow Q'$ and $P' \mid \varphi \mathcal{R} Q'$.

Weak bisimilarity, written $\approx$, is the greatest weak bisimulation.

Clause 1 requires not only that Q catches up entailment $P \rhd \varphi$ with $Q \Rightarrow Q'$ and $Q' \rhd \varphi$ for some Q', but also that Q' is related to P in the bisimulation candidate. This is necessary, since otherwise we could prove $\varphi \mid a \oplus b \approx (\varphi \mid a) \oplus (\varphi \mid b)$ where $\oplus$ is the internal choice operator, whereas those two processes are distinguished by the context $[\cdot] \mid [\varphi]\tau.(\overline{a} \mid \overline{b})$.

Note that iterating clause 1, and using the fact that conditions are persistent, we get that if $\mathcal{R}$ is a weak bisimulation and if $P \mathcal{R} Q$ then there is a Q' such that $P \mathcal{R} Q'$ and $Q \Longrightarrow Q'$ with $\Phi(P) \subseteq \Phi(Q')$. Iterating this remark (on P, then on Q', and so on, as long as $\Phi(P) \neq \Phi(Q')$—which terminates since the number of entailed conditions that are not reflexive is bounded[5]) we can prove that if $P \mathcal{R} Q$ then for some P' and Q', we have $P' \mathcal{R} Q'$ with $P \Longrightarrow P'$ and $Q \Longrightarrow Q'$ such that $\Phi(P') = \Phi(Q')$.

Clause 3 could have been of the form $Q \stackrel{[\varphi]\tau}{\Longrightarrow} Q'$ (or $Q \Longrightarrow Q'$ when φ is reflexive), but this would have been too strong a requirement since Q must be able to answer with no transition at all. The problem is revealed when we want the law $\tau \approx 0$: τ can make a $[\varphi]\tau$ transition, whereas 0 cannot.

To reach a characterisation of $\cong_{\mathrm{bn}}$, the soundness of bisimilarity seems achievable and does not depend on which sub-calculus we are considering. On the contrary, to show completeness, we need to use some tester processes. To test visible transitions (clause 2) and those with labels of the form $[\varphi]\tau$ (clause 3), the usual tester processes are enough. To test clause 1, of course, the tester process $[\varphi]\tau.(\overline{w} \mid w)$ is enough to conclude the proof and reach a characterisation $(\approx) = (\cong_{\mathrm{bn}})$.

However, if there is no prefix $[\varphi]\tau$ in the calculus considered, the situation is more complicated[6]. Suppose that we have now a replication operator, and consider process M below:

$$M \triangleq !(\boldsymbol{\nu} x)(ax \mid \overline{b}x) \mid !(\boldsymbol{\nu} x)(\overline{a}x \mid bx) \ .$$

We can prove that $M \cong_{\mathrm{bn}} M \mid a \curlyvee b$, whereas there is no M' for which $M \Longrightarrow M' \rhd a \curlyvee b$. This problem was in fact already present in the explicit fusion calculus ($M \cong_{\mathrm{ef}} M \mid a{=}b$), but not in implicit fusions. Indeed the equivalent of $a{=}b$ is $(u)(\overline{u}a \mid ub)$ for which there is no special clause in the bisimulation. And indeed, there is a weak behavioural theory for implicit fusions, together with the corresponding axiomatisation [PV98b]. This motivates an implicit presentation of πP, which we discuss below.

Implicit πP We do not see any natural "implicit" version of πP, mimicking the relation between explicit fusions and implicit fusions, whereby the extension of the preorder along a communication would not generate an arc process. One approach—that might have worked if one wanted to derive the implicit fusion calculus from explicit fusions—would be to require at least to have some processes $f(P)$ and $g(P)$ such that:

$$(\boldsymbol{\nu} a)(ca \mid \overline{c}b.P) \longrightarrow f(P) \qquad\qquad (\boldsymbol{\nu} a)(\overline{c}a \mid cb.P) \longrightarrow g(P) \ ,$$

where f and g should be compositional functions satisfying the following equations:

$$f(P) \sim (\boldsymbol{\nu} a)(b/a \mid P) \qquad\qquad g(P) \sim (\boldsymbol{\nu} a)(a/b \mid P) \ .$$

This would imply in particular (using in particular the expansion law and law L25):

$$f(a) \sim b \qquad\qquad f(\overline{\{a\}}) \sim 0 \qquad\qquad f(a \mid \overline{\{a\}}) \sim b + \tau \not\sim f(a) \mid f(\overline{\{a\}}) \ .$$

Similarly, $g(a \mid \overline{a}) \not\sim g(a) \mid g(\overline{a})$. Therefore, it is not straightforward to give a compositional description of $f(P \mid Q)$ and $g(P \mid Q)$ in terms of P and Q, because of the possible interactions between P and Q on channel a.

[5] Indeed when $P \Longrightarrow P'$ we have $|\Phi(P')^{\neq}| \leq |\mathrm{fn}(P')|^2 \leq |\mathrm{fn}(P)|^2$. Note that visible transitions $\stackrel{\alpha(x)}{\longrightarrow}$ create new names, so in general in a bisimulation, these sets are not bounded (if we add the replication operator).

[6] In the strong case, $[\varphi]\tau$ is not necessary: it is enough to install a tester of the form $\overline{\alpha}.\overline{w_1} \mid \beta.\overline{w_2}$ such that $\varphi = \alpha \curlyvee \beta$ and to count the number of reductions before both $\overline{w_1}$ and $\overline{w_2}$ appear as barbs

Reversible eager reductions One of the inconveniences of eager reduction is the commitments it performs, as seen in the beginning of Section 4.3. This prevents the hope of implementing the more compelling by-need semantics using smaller-steps eager reductions, as shown in Example 4.3.10. One way would be to try to revert these eager reductions, exploiting ideas from backtracking in reversible process calculi [DK04] (note that we are in a much simpler setting, as we only need to backtrack rewritings, and not communications).

Indeed, at the cost of a more convoluted description of processes, there is a mean to decompose by-need reduction into reversible small-step reductions. We extend the grammar of the subject of prefixes to *sets* of extended names, as follows:

$$\begin{array}{rcl} \alpha & ::= & a \mid \{a\} \\ \varepsilon & ::= & \{\alpha_1, \ldots, \alpha_n\} \\ P & ::= & 0 \mid P \mid Q \mid (\boldsymbol{\nu} a)P \mid \overline{\varepsilon}\langle b\rangle.P \mid \varepsilon\langle b\rangle.P \mid a\not b \end{array}$$

then we would have a new notion of reduction (we omit the output counterpart of the first two laws):

$$\frac{a \in \varepsilon^+}{b\not a \mid \varepsilon\langle c\rangle.P \longrightarrow b\not a \mid (\varepsilon \cup \{b\})\langle c\rangle.P} \qquad \frac{a \in \varepsilon^-}{a\not b \mid \varepsilon\langle c\rangle.P \longrightarrow a\not b \mid (\varepsilon \cup \{\{b\}\})\langle c\rangle.P}$$

$$\frac{\varepsilon_1^+ \cap \varepsilon_2^- \neq \emptyset}{\overline{\varepsilon_1}\langle b\rangle.P \mid \varepsilon_2\langle c\rangle.Q \longrightarrow b\not c \mid P \mid Q} \qquad \frac{\varepsilon_1^- \cap \varepsilon_2^+ \neq \emptyset}{\overline{\varepsilon_1}\langle b\rangle.P \mid \varepsilon_2\langle c\rangle.Q \longrightarrow b\not c \mid P \mid Q}$$

where $\varepsilon^+ \triangleq \{a \mid a \in \varepsilon\}$ and $\varepsilon^- \triangleq \varepsilon^+ \cup \{a \mid \{a\} \in \varepsilon\}$.

The first two rules are "careful" rewriting rules, that leave the possibility for other arcs to intervene on the prefix and increase the corresponding set. The last two rules tell us that a reduction is possible when the two sets in subject positions intersect (we do not allow a reduction between two protected prefixes, as it is the case in by-need).

Such a reduction would indeed never make commitments, except when doing by-need reductions. If $P \longrightarrow P'$ then we would have either $P \simeq_{\mathrm{bn}} P'$ or $P \longrightarrow_{\mathrm{bn}} P'$. Moreover any $\longrightarrow_{\mathrm{bn}}$ reduction can be decomposed into one $\longrightarrow$ reduction or more. We believe that the corresponding weak bisimilarity would coincide with $\approx_{\mathrm{bn}}$.

Note that this would have an interest only at an implementation level—and not for expressiveness or for discriminative power—as those prefixes can be encoded using sums:

$$\varepsilon\langle b\rangle.P \sim \sum_{\alpha \in \varepsilon} \alpha\langle b\rangle.P \ .$$

Note how with this interpretation, the first two reduction rules correspond to prefix laws in the axiomatisation of πP, Table 4.2.

Chapter 5

Bisimulation enhancements for higher-order languages

Previous chapters showed that bisimulation, when proved sound, is an efficient proof method to capture barbed congruences for a given calculus $\mathcal{L}$: to prove $P \simeq_{\mathcal{L}} Q$, one exhibits a relation containing the pair (P, Q) and then proves it to be a bisimulation. This proof method can be enhanced by employing relations called *'bisimulations up to'* [Len98, PS12, RBR13]. These need not be bisimulations; they are simply *contained in* a bisimulation. Such techniques have been widely used in languages for mobility such as the π-calculus or higher-order languages such as the λ-calculus, or Ambients (e.g., [SW01, MN05, Las98a], as well as in Section 4.5).

This chapter contributes to improve these techniques and their usability. It is independent from Chapters 3 and 4, apart from Section 5.2.2.4.

Several forms of bisimulation enhancements have been introduced: 'bisimulation up to bisimilarity' [Mil89] where the derivatives obtained when playing bisimulation games can be rewritten using bisimilarity itself; 'bisimulation up to transitivity' where the derivatives may be rewritten using the up-to relation; 'bisimulation up to context' [San98], where a common context may be removed from matching derivatives. Further enhancements may exploit the peculiarities of the definition of bisimilarity on certain classes of languages: e.g., the up to injective substitution techniques of the π-calculus [JR99, SW01], techniques for shrinking or enlarging the environment in languages with information hiding mechanisms (e.g., existential types, encryption and decryption constructs [AG98, SP07b, SP07a]), frame equivalence in the psi-calculi [PP14], or higher-order languages [Las98b, KW06]. Lastly, it is important to notice that one often wishes to use *combinations* of up-to techniques. For instance, up to context alone does not appear to be very useful; its strength comes out in association with other techniques, such as up to bisimilarity or up to transitivity. For example, in Section 4.5 up to bisimilarity and up to bisimilarity and restriction are used (Definitions 4.5.27 and 4.5.31), but up to restriction alone is not.

The main problem with up-to techniques is proving their soundness (i.e. ensuring that any 'bisimulation up to' is contained in bisimilarity). In particular, the proofs of complex combinations of techniques can be difficult or, at best, long and tedious. And if one modifies the language or the up-to technique, the entire proof has to be redone from scratch. Indeed the soundness of some up-to techniques is quite fragile, and may break when such variations are made. For instance, in certain models up to bisimilarity may fail for weak bisimilarity, and in certain languages up to bisimilarity and context may fail even if bisimilarity is a congruence relation and is strong (treating internal moves as any other move).

This problem has been the motivation for the development of a theory of enhancements, summarised in [PS12]. Expressed in the general theory of fixed points in complete lattices, this theory has been fully developed for both strong and weak bisimilarity, in the case of first-order labelled transition systems (LTSes) where labels are simple syntactic objects. In this framework, up-to techniques are represented using *compatible* functions, whose class enjoys nice algebraic properties. This allows one to derive complex up-to

techniques algebraically, by composing simpler techniques by means of a few operators.

Only a small part of the theory has been transported onto other forms of transition systems, on a case by case basis, for instance those needed in the π-calculus, or in higher-order formalisms. These techniques have been developed in a 'by-need' fashion: one has concrete bisimilarity proofs to carry out, and ensures the soundness of the up-to techniques employed in such proofs. Transferring the whole theory of bisimulation enhancements outside the realm of first-order transition systems would be a substantial and non-trivial effort. Moreover it might have limited applicability, as this work would probably have to be based on specific shapes for transitions and bisimilarity (a wide range of variations exist, e.g., in higher-order languages).

This chapter tackles this problem. We explore here a different approach to the transport of the theory of bisimulation enhancements onto richer languages. The approach consists in exhibiting fully abstract translations of the more sophisticated LTSes and bisimilarities onto first-order LTSes and bisimilarity. This allows us to import directly the existing theory for first-order bisimulation enhancements onto the new languages. Most importantly, the schema allows us to combine up-to techniques for the richer languages. The only additional ingredient that has to be provided manually is the soundness of some up-to techniques that are specific to the new languages. This typically includes the up to context techniques, since those contexts are not first-order.

We consider four languages: the π-calculus, the call-by-name λ-calculus, an imperative call-by-value λ-calculus (a call-by-value λ-calculus with references), and the Higher-Order π-calculus.

We moreover focus on weak bisimilarity, since its theory is more delicate than that of strong bisimilarity (e.g., the congruence properties) and these differences are magnified in theories of enhancements of the bisimulation proof method. When we translate a transition system into a first-order one, the grammar for the labels can be complex (e.g. include terms, labels, or contexts). What makes the system 'first-order' is that labels are taken as syntactic atomic objects, that may only be checked for syntactic equality. Note that full abstraction of the translation does not imply that the up-to techniques come for free: further conditions must be ensured. We shall see this with the π-calculus, where early bisimilarity can be handled (Section 5.2.1) but not the late one (Section 5.2.2.2).

Forms of up to context have already been derived for the languages we consider [Las98a, SW01, SKS11]. The corresponding soundness proofs are difficult (especially in λ-calculi), and require a mix of induction (on contexts) and coinduction (to define bisimulations). Recasting up to context within the theory of bisimulation enhancements has several advantages. First, this allows us to combine this technique with other techniques, directly. Second, substitutivity (or congruence) of bisimilarity becomes a corollary of the compatibility of the up to context function (in higher-order languages these two kinds of proofs are usually hard and very similar). And third, this allows us to decompose the up to context function into smaller pieces, essentially one for each operator of the language, yielding more modular proofs, also allowing, if needed, to rule out those contexts that do not preserve bisimilarity (e.g., input prefix in the π-calculus).

Section 5.1 recalls the ingredients we need from the theory of bisimulation enhancements, together with some other basic tools we introduce for the remaining of the chapter. Section 5.2 presents the translation of the π-calculus capturing early bisimilarity, with the corresponding up to context techniques. We also give an account for other bisimilarities. Section 5.3 is the translation of the call-by-name λ-calculus, through environmental bisimilarity, again with up to context techniques. Section 5.4 does the same for the imperative λ-calculus and provides examples of bisimulations up to using combinations of up-to techniques. Section 5.5 gives, with fewer details, LTSes for some other calculi.

5.1 Background: a theory of bisimulation enhancements

This section is largely inspired by Chapter 6 of [PS12], which is an evolution from [Pou08], the latter also handling a more abstract theory. Most proofs in this section are rather easy and abstracted to a lattice-theoretic level.

To clarify the meaning of 'first order', we give a more formal definition of *first-order Labelled Transition System*, or simply LTS. An LTS a triple $(Pr, Act, \longrightarrow)$ where Pr is a non-empty set of *processes*, Act is the set of

actions (or *labels*), and $\longrightarrow \subseteq \mathit{Pr} \times \mathit{Act} \times \mathit{Pr}$ is the *transition relation*. As usual, we use P, Q, R to range over the processes of the LTS, μ over the labels in *Act*, $\mathcal{R}$ and $\mathcal{S}$ over relations on *Pr*, and write $P \xrightarrow{\mu} Q$ when $(P, \mu, Q) \in \longrightarrow$. We assume that *Act* includes a special action τ that represents an internal activity of the processes. We derive bisimulation from the notion of *progression* between relations:

Definition 5.1.1 (Progression)**.** Relation $\mathcal{R}$ *progresses to* relation $\mathcal{S}$, written $\mathcal{R} \rightarrowtail \mathcal{S}$, if

- whenever $P \xrightarrow{\mu} P'$ there is Q' s.t. $Q \xrightarrow{\mu} Q'$ and $P' \; \mathcal{S} \; Q'$;
- whenever $Q \xrightarrow{\mu} Q'$ there is P' s.t. $P \xrightarrow{\mu} P'$ and $P' \; \mathcal{S} \; Q'$.

Figure 5.1: $\mathcal{R}$ progresses to $\mathcal{S}$ ($\mathcal{R} \rightarrowtail \mathcal{S}$)

Definition 5.1.1 can be depicted in diagrams: $\mathcal{R} \rightarrowtail \mathcal{S}$ if the relations satisfy the diagrams of Figure 5.1. Clearly, $\mathcal{R}$ is a bisimulation if and only if $\mathcal{R} \rightarrowtail \mathcal{R}$. Alternatively, we define an equivalent notion in terms of a function on relations:

Definition 5.1.2 (Progression operator)**.** The *progression operator* $\mathbf{b}$ is the following function on relations on processes:

$$\mathbf{b} : \mathcal{S} \mapsto \mathcal{S} \cap \bigcup_{\mathcal{R} \rightarrowtail \mathcal{S}} \mathcal{R} \ .$$

In yet other words, bisimulations are relations $\mathcal{R}$ such that $\mathcal{R} \subseteq \mathbf{b}(\mathcal{R})$. Consequently, bisimulations are the post-fixpoints of $\mathbf{b}$, which is a monotone function. The proof of the Knaster–Tarski theorem on the powerset lattice then tells us that the union of post-fixpoints of $\mathbf{b}$ is the greatest fixpoint of $\mathbf{b}$.

Writing $\mathsf{gfp}(f)$ for the join of the post-fixpoints of a monotone function f, we know that $\mathsf{gfp}(f)$ is the greatest fixpoint of f, which trivially implies that $\mathsf{gfp}(f)$ itself is a post-fixpoint of f. Instantiating f with $\mathbf{b}$, the latter states that bisimilarity is a bisimulation. Summing up, we have:

$$\mathsf{gfp}(f) \triangleq \bigcup_{\mathcal{R} \subseteq f(\mathcal{R})} \mathcal{R} \qquad\qquad \sim \triangleq \mathsf{gfp}(\mathbf{b}) \qquad\qquad \sim \subseteq \mathbf{b}(\sim) \ .$$

The equivalence of interest often is weak bisimilarity instead of strong bisimilarity. Instead of making similar proofs for the two cases, we handle both of them in a uniform manner: we generalise $\mathbf{b}$ to any monotone function b, proving results for other instances[1] of b (one of which corresponding to weak bisimilarity):

Definition 5.1.3 (b-simulation)**.** We say a relation $\mathcal{R}$ is a *b-simulation* if $\mathcal{R} \subseteq b(\mathcal{R})$. We call *$b$-similarity* the relation $\mathsf{gfp}(b)$, the union of all b-simulations.

We already have seen that the $\mathbf{b}$-simulations are the bisimulations, and that bisimilarity is $\mathbf{b}$-similarity $\mathsf{gfp}(\mathbf{b})$. When referring to general b-simulations, and not $\mathbf{b}$-simulations, we will simply write 'simulation', and similarly for similarity.

By definition, simulation is a proof method for similarity: $\mathcal{R} \subseteq b(\mathcal{R})$ implies $\mathcal{R} \subseteq \mathsf{gfp}(b)$. Unfortunately, for some relations $\mathcal{R}$, proofs of $\mathcal{R} \subseteq b(\mathcal{R})$ contain redundancies. Those redundancies may sometimes be avoided by replacing $b(\mathcal{R})$ with a bigger relation $b(f(\mathcal{R}))$, where function f is what we call an up-to technique. This weaker requirement corresponds to being a post-fixpoint of $b \circ f$, where $\circ$ denotes functional composition.

[1] Going further, Pous [Pou08] has abstracted, when possible, the powerset lattice of relations to any complete lattice, deriving techniques for coinduction itself.

Definition 5.1.4 (b-simulation up to)**.** A *b-simulation up to f* is a $(b \circ f)$-simulation.

We want for 'simulation up to' to be a proof method for similarity, i.e. we want that b-simulations up to f are included in b-similarity. This is equivalent to saying that $(b \circ f)$-similarity is also included in b-similarity. Hence, the desired property for techniques—called soundness—is a simple inclusion:

Definition 5.1.5 (Soundness)**.** We say f is b-sound if $\mathsf{gfp}(b \circ f) \subseteq \mathsf{gfp}(b)$.

As discussed above, this definition of soundness implies that

$$\forall \mathcal{R} \quad \mathcal{R} \subseteq b(f(\mathcal{R})) \;\Rightarrow\; \mathcal{R} \subseteq \mathsf{gfp}(b) \;, \tag{5.1}$$

which is the proof method that we will use in the following. For (5.1) to imply that f is b-sound, we only need for $\mathsf{gfp}(b \circ f)$ to be a post-fixpoint of $b \circ f$, which is true when $b \circ f$ is monotone. This monotonicity condition is satisfied for most functions f and b of interest, including $\mathbf{b}$ and all the techniques of this chapter.

In the particular case of $b = \mathbf{b}$, bisimulations up to f are relations $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail f(\mathcal{R})$. This means that those relations satisfy the "bisimulation up to" diagrams in Figure 5.2. If $f(\mathcal{R})$ contains $\mathcal{R}$, satisfying those diagrams is easier than satisfying those for plain bisimulation (Figure 1.1). However, $\mathcal{R} \rightarrowtail f(\mathcal{R})$ does not necessarily imply that $\mathcal{R} \subseteq \mathbf{b}(f(\mathcal{R}))$ since the latter requires in addition that $\mathcal{R} \subseteq f(\mathcal{R})$, which might not be satisfied—for example by functions in Section 5.1.1. This technicality is of little concern: if $\mathcal{R} \rightarrowtail f(\mathcal{R})$ then $\mathcal{R} \subseteq \mathbf{b}(g(\mathcal{R}))$ with $g : \mathcal{R} \mapsto \mathcal{R} \cup f(\mathcal{R})$.

Figure 5.2: $\mathcal{R}$ is a bisimulation up to f (i.e. $\mathcal{R} \rightarrowtail f(\mathcal{R})$)

In general, soundness is not closed under composition, pointwise union of functions (see Chapter 6.3.3 of [PS12]), or other algebraic operators. As a remedy to this, a stronger notion, compatibility, is introduced ($\subseteq$ is lifted to functions: $(f \subseteq g) \triangleq (\forall \mathcal{R}\ f(\mathcal{R}) \subseteq g(\mathcal{R}))$):

Definition 5.1.6 (Compatibility)**.** A monotone function f is *b-compatible* if $f \circ b \subseteq b \circ f$.

As desired, the set of compatible functions is a subset of that of sound functions:

Proposition 5.1.7. *Suppose f and b are monotone. Then if f is b-compatible then f is b-sound.*

Some b-sound functions are not b-compatible, but the class is expressive enough to include most common up-to techniques, and enjoys pleasant algebraic properties. Simple examples of compatible functions are the identity function and the function mapping any relation onto a simulation. The class of compatible functions is closed under function composition and union, and thus under omega-iteration.

Proposition 5.1.8. *Suppose b is a monotone function. The following functions are b-compatible:*

1. *the identity function* $\mathsf{id} : \mathcal{R} \mapsto \mathcal{R}$*,*
2. *the constant function $\mathcal{R} \mapsto \mathcal{S}$ when $\mathcal{S}$ is a b-simulation,*
3. *$f \circ g$ when f and g are monotone and b-compatible,*
4. *$\bigcup F : \mathcal{R} \mapsto \bigcup_{f \in F} f(\mathcal{R})$ when F is a set of b-compatible functions,*
5. *$f^\omega : \mathcal{R} \mapsto \bigcup_{n \in \mathbb{N}} f^n(\mathcal{R})$ when f is monotone and b-compatible.*

A pleasant consequence of compatibility is the substitutivity property:

Lemma 5.1.9. *Suppose f and b are monotone. Then if f is b-compatible then $f(\mathsf{gfp}(b)) \subseteq \mathsf{gfp}(b)$.*

In the particular case where $b = \mathbf{b}$ and f is a closure operation, Lemma 5.1.9 states that bisimilarity is closed by f. This is especially useful if f is the context closure function $\mathcal{C}$: the compatibility of $\mathcal{C}$ implies both the soundness of any combination of the up to context technique with other compatible functions, and the congruence of bisimilarity. This considerably reduces the proof effort, getting two important properties with one, typically short, proof. Indeed, proving compatibility is usually straightforward, when the result holds (there are not a lot of choices, compared to soundness proofs). A good example is the environmental bisimilarity in λ-calculi, for which proofs of soundness of up to context techniques have lengthy common parts with proofs of congruence [SKS11].

Proofs of compatibility themselves contain case analyses that one would like to split into several independent results. Unfortunately, they often depend on each other. The definition of compatibility can be refined even further to take this into account:

Definition 5.1.10 (Evolution)**.** We say that f evolves to g through b, written $f \overset{b}{\rightsquigarrow} g$, when $f \circ b \subseteq b \circ g$.

One can read $\overset{b}{\rightsquigarrow}$ as a dependency: if g is compatible and $f \overset{b}{\rightsquigarrow} g$ then f would be included into a compatible function ($f \cup g$ where $\cup$ is lifted to pointwise union), which would imply the desired properties of soundness and substitutivity. Since evolution behaves well with respect to union and composition, and compatibility corresponds to a function evolving to itself, these "dependencies" can have cycles. This is clarified and formalised in the following lemma:

Lemma 5.1.11. *Suppose b is monotone, and F is a set of monotone functions. Let $u \triangleq \bigcup F$. Suppose that for all f in F we have $f \overset{b}{\rightsquigarrow} u^\omega$. Suppose that u^ω is idempotent. Then u^ω is b-compatible.*

Proof. We know that $f \circ b \subseteq b \circ u^\omega$ for all $f \in F$, hence $\bigcup_{f \in F}(f \circ b) \subseteq b \circ u^\omega$, which we can rewrite to $u \circ b \subseteq b \circ u^\omega$. From this, we prove by induction on n that $u^n \circ b \subseteq b \circ u^\omega$. The case $n = 0$ ($b \subseteq b \circ u^\omega$) follows from the monotonicity of b. Suppose now $u^n \circ b \subseteq b \circ u^\omega$. We consider then $u^{n+1} \circ b$:

$$\begin{array}{rcll} u \circ u^n \circ b & \subseteq & u \circ b \circ u^\omega & \text{by induction and monotonicity of } u \\ & \subseteq & b \circ u^\omega \circ u^\omega & \text{by hypothesis} \\ & \subseteq & b \circ u^\omega & \text{by idempotence of } u^\omega. \end{array}$$

This concludes the induction. We can deduce from this that $\bigcup_{n \in \mathbb{N}}(u^n \circ b) \subseteq b \circ u^\omega$ which we rewrite to $u^\omega \circ b \subseteq b \circ u^\omega$, hence u^ω is b-compatible. □

Remark 5.1.12 (Conditions for idempotence)**.** The condition of u^ω being idempotent ($u^\omega \circ u^\omega = u^\omega$) is equivalent to $u \circ u^\omega \subseteq u^\omega$. When proving this inclusion in a lattice-theoretic setting, one would like to have the hypothesis that each function f in F preserves joins (i.e. unions). Unfortunately, in a powerset lattice, those functions are degenerate: since every subset is the union of its singletons, a join-preserving function f can be described in a pointwise manner:

$$f(\mathcal{R}) = f\Big(\bigcup_{x \in \mathcal{R}} \{x\}\Big) = \bigcup_{x \in \mathcal{R}} f(\{x\}) \ .$$

Indeed, there are important compatible functions that *do not* preserve joins. In fact, a weaker equation makes for a better condition. We say a function f on powersets is *continuous* if it is characterised by its image on finite subsets, i.e.:

$$f(\mathcal{R}) = \bigcup_{X \in \mathcal{P}_{\mathsf{fin}}(\mathcal{R})} f(X) \tag{5.2}$$

where $\mathcal{P}_{\mathsf{fin}}(\mathcal{R})$ is the set of finite subsets of $\mathcal{R}$. This is exactly equivalent to f being Scott-continuous in the subset partial order in a powerset. (Scott continuity is usually formulated in other terms [AC98], but in

powersets, we prefer (5.2) which is more obvious to check.) The class of continuous functions is closed under composition, intersection, arbitrary union and—when the powerset is the set of relations—under relational composition. In addition, if f is continuous, then f is monotone and f^ω is idempotent.

Another formulation of (5.2) is that $(P,Q) \in f(\mathcal{R})$ iff there are pairs $(P_1,Q_1) \in \mathcal{R}, \ldots, (P_n,Q_n) \in \mathcal{R}$ such that $(P,Q) \in f(\{(P_1,Q_1),\ldots,(P_n,Q_n)\})$. Using this formulation, it is clear that functions with inductive definitions of the form:

"if $P_1 \mathcal{R} Q_1$ and $P_2 \mathcal{R} Q_2$ then $(P_1 \mid P_2)\ f(\mathcal{R})\ (Q_1 \mid Q_2)$"
"if $P \mathcal{R} Q$ then for all R we have $(P \mid R)\ f(\mathcal{R})\ (Q \mid R)$"

are continuous, as most of the functions we are considering in this chapter. We see that the number of "requests" on $\mathcal{R}$ is bounded for a given pair (P,Q). Continuity does not, however, authorise an infinite number of such requests (this could make f^ω not idempotent). For example, the function defined by

"if $\forall R \in A\ (P \mid R)\ \mathcal{R}\ (Q \mid R)$ then $P\ f(\mathcal{R})\ Q$"

is not continuous if A is infinite (it is if A is finite for all P and Q).

Continuity allows us to find the compatible function with a relatively simple expression: if $f \overset{b}{\rightsquigarrow} (\cup F)^\omega$ then the b-compatible function containing all fs is $(\cup F)^\omega$, assuming the other hypotheses of Lemma 5.1.11. This is interesting, as $(\cup F)^\omega$ coincides with interesting functions (for example, context closure). If we do not care about the shape of the compatible function, we can drop the idempotence condition, and know that f is included in *some* compatible function, obtained by an omega-iteration of order two.

Omega-iteration being a function on functions on relations, we say informally that it is a function "of order two". We can reason about such functions: the following lemma shows that omega-iteration is "compatible" in a second-order sense, which simply means that it preserves evolutions.

Lemma 5.1.13. *If f and b are monotone and $f \overset{b}{\rightsquigarrow} g$, then $f^\omega \overset{b}{\rightsquigarrow} g^\omega$.*

Proof. To prove $f^\omega \circ b \subseteq b \circ g^\omega$, it is enough to prove $f^n \circ b \subseteq b \circ g^\omega$ for all n. Since $b \circ g^n \subseteq b \circ g^\omega$ (by monotonicity of b) we prove only the following:

$$f^n \circ b \subseteq b \circ g^n \quad . \tag{5.3}$$

We prove (5.3) by induction on n. The case $n = 0$ (i.e. $b \subseteq b$) is trivial. Suppose (5.3) for n, we prove it for $n+1$. By monotonicity of f, (5.3) implies $f \circ (f^n \circ b) \subseteq f \circ (b \circ g^n)$, and $(f \circ b) \circ g^n \subseteq (b \circ g) \circ g^n$ by hypothesis, which concludes the induction. □

Lemma 5.1.14. *If b and f are monotone and $f \overset{b}{\rightsquigarrow} f^\omega$ then $f \subseteq g \overset{b}{\rightsquigarrow} g$ for some monotone function g.*

Proof. We take for g the monotone function $\underline{f}$, where

$$\underline{f} \triangleq \bigcup_{n \in \mathbb{N}} f_n \qquad\qquad f_0 \triangleq f \qquad\qquad f_{n+1} \triangleq (f_n)^\omega \quad ,$$

and we show that $\underline{f}$ is b-compatible, i.e. $\underline{f} \circ b \subseteq b \circ \underline{f}$, for which it is enough to show that $f_n \circ b \subseteq b \circ \underline{f}$ for all n. By monotonicity of b, we prove in fact that $f_n \circ b \subseteq b \circ f_{n+1}$ (i.e. $f_n \overset{b}{\rightsquigarrow} f_{n+1}$) by induction on n.

- $f_0 \overset{b}{\rightsquigarrow} f_1$ is $f \overset{b}{\rightsquigarrow} f^\omega$ (our hypothesis).
- if $f_n \overset{b}{\rightsquigarrow} f_{n+1}$ then $f_n^\omega \overset{b}{\rightsquigarrow} f_{n+1}^\omega$ by Lemma 5.1.13 (f_n and b are monotone) i.e. $f_{n+1} \overset{b}{\rightsquigarrow} f_{n+2}$.

Note that each f_n is monotone since $\cdot^\omega$ preserves monotonicity, and hence $\underline{f}$ is monotone as well. □

Lemma 5.1.11 enables some modularity in proofs of compatibility. It is especially useful for Section 5.1.1, where the context closure function $\mathcal{C}$ is decomposed into a set of functions each corresponding to an operator, and $\mathcal{C} = u^\omega$ is indeed idempotent, and then $\mathcal{C}$ is itself compatible. Lemma 5.1.14 gets rid of the idempotence condition, with a bigger and more complex compatible function. This does not matter much, since when $f \subseteq g$ and $g \overset{b}{\rightsquigarrow} g$, since g is sound and substitutive, then f is, too:

$$\mathsf{gfp}(b \circ f) \subseteq \mathsf{gfp}(b \circ g) \underset{\text{Prop. 5.1.7}}{\subseteq} \mathsf{gfp}(b) \qquad\qquad f(\mathsf{gfp}(b)) \subseteq g(\mathsf{gfp}(b)) \underset{\text{Prop. 5.1.8}}{\subseteq} \mathsf{gfp}(b) \ ,$$

since b and $\mathsf{gfp}(\cdot)$ are monotone.

This reasoning allows us to reason up to some function, when establishing compatibility of other functions. When F is a set of functions, we say that F is *b-compatible up to* if for all f in F, it holds that $f \overset{b}{\rightsquigarrow} (g \cup (\cup F))^\omega$ for a function g that has already been proven b-compatible. We sometimes say that F is *b-compatible up to g*, to specify which compatible function is employed. When we can, we prove that the omega-iteration is idempotent. Lemma 5.1.7 and 5.1.8 remain valid when 'f is b-compatible' is replaced by '$f \in F$ and F is b-compatible up to'.

Remark 5.1.15. In Definition 5.1.2, $\mathbf{b}(\mathcal{S})$ is defined as the intersection of $\mathcal{S}$ with the union of all relations progressing to $\mathcal{S}$, instead of just the union. Considering only the union (of all relations progressing to $\mathcal{S}$) would give us the same post-fixpoints (the same bisimulations), but the actual definition of $\mathbf{b}$ gives us more compatible functions: to prove that f is $\mathbf{b}$-compatible, we have to get $f(\mathcal{R}) \rightarrowtail f(\mathcal{S})$ knowing not only that $\mathcal{R} \rightarrowtail \mathcal{S}$, but also with the additional hypothesis $\mathcal{R} \subseteq \mathcal{S}$, which is important when some transitions emanating from $f(\mathcal{R})$-related processes do not correspond to transitions emanating from processes in $\mathcal{R}$. (We also have to prove $f(\mathcal{R}) \subseteq f(\mathcal{S})$ but f has to be monotone anyway.)

For example, take $a(\mathcal{R}) = \{(a.P, a.Q) \mid P \ \mathcal{R} \ Q\}$ (note that $a(\mathcal{R}) \rightarrowtail \mathcal{R}$) and $c(\mathcal{R}) = a(\mathcal{R}) \cup \mathcal{R}$. Then c is compatible: if $\mathcal{R} \rightarrowtail \mathcal{S}$ and $\mathcal{R} \subseteq \mathcal{S}$ then $c(\mathcal{R}) \rightarrowtail \mathcal{R} \cup \mathcal{S} = \mathcal{R}$. The hypothesis $\mathcal{R} \subseteq \mathcal{S}$ is necessary, as:

$$\begin{array}{rccccc} & \{(b,b)\} & \rightarrowtail & \{(0,0)\} & \rightarrowtail & \emptyset \ , \\ \text{but} & c(\{(b,b)\}) & \not\rightarrowtail & c(\{(0,0)\}) & \not\rightarrowtail & c(\emptyset) \ . \end{array}$$

In passing, we have $a \overset{\mathbf{b}}{\rightsquigarrow} \mathsf{id} \overset{\mathbf{b}}{\rightsquigarrow} \mathsf{id}$, and a is $\mathbf{b}$-compatible up to id.

Weak bisimilarity We now instantiate b with a different monotone function $\mathbf{wb}$ such that $\mathbf{wb}$-simulations are weak bisimulations. We recall from Definition 2.1.8 that $\overset{\hat{\mu}}{\Longrightarrow}$ is $\Longrightarrow = \overset{\tau}{\longrightarrow}^*$ if $\mu = \tau$ and $\Longrightarrow\overset{\mu}{\longrightarrow}\Longrightarrow$ otherwise. *Weak progression*, $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$, and *weak bisimilarity*, $\approx$, are obtained from Definition 5.1.1 and Definition 5.1.2 by allowing the processes to answer using $\overset{\hat{\mu}}{\Longrightarrow}$ rather than $\overset{\mu}{\longrightarrow}$.

Definition 5.1.16 (Weak progression)**.** For relations $\mathcal{R}, \mathcal{S}$ on the processes of an LTS, we say that $\mathcal{R}$ *weakly progresses to* $\mathcal{S}$, written $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$, if $P \ \mathcal{R} \ Q$ implies:

- whenever $P \overset{\mu}{\longrightarrow} P'$ there is Q' s.t. $Q \overset{\hat{\mu}}{\Longrightarrow} Q'$ and $P' \ \mathcal{S} \ Q'$;
- whenever $Q \overset{\mu}{\longrightarrow} Q'$ there is P' s.t. $P \overset{\hat{\mu}}{\Longrightarrow} P'$ and $P' \ \mathcal{S} \ Q'$.

The weak progression operator $\mathbf{wb}$ is defined as follows:

$$\mathbf{wb} : \mathcal{S} \mapsto \mathcal{S} \cap \bigcup_{\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}} \mathcal{R} \ .$$

Again, $\mathcal{R}$ is a weak bisimulation iff $\mathcal{R} \subseteq \mathbf{wb}(\mathcal{R})$, and weak bisimilarity indeed coincides with the union of all weak bisimulations: $\approx \ = \mathsf{gfp}(\mathbf{wb})$.

Certain closure properties for compatible functions hold in the strong case (i.e. for $\mathbf{b}$), but not for the weak case (i.e. for $\mathbf{wb}$). The main example is the *chaining operator* $\frown$, which implements relational composition:

$$f \frown g\ (\mathcal{R}) \triangleq f(\mathcal{R})\ g(\mathcal{R})$$

where $f(\mathcal{R})\ g(\mathcal{R})$ stands for the composition of the two relations $f(\mathcal{R})$ and $g(\mathcal{R})$. Using chaining we can obtain the $\mathbf{b}$-compatibility of the function 'up to transitivity' mapping any relation $\mathcal{R}$ onto its reflexive and transitive closure $\mathcal{R}^\star$.

In contrast, in the weak case bisimulation up to bisimilarity ($\mathcal{R} \mapsto \approx\mathcal{R}\approx$) is unsound, whereas strong bisimulation up to bisimilarity ($\mathcal{R} \mapsto \sim\mathcal{R}\sim$) is $\mathbf{b}$-compatible. This is a major drawback in up-to techniques for weak bisimilarity, which can be partially overcome by resorting to the *expansion* relation $\gtrsim$ [AKH92]. Expansion is an asymmetric refinement of weak bisimilarity whereby $P \gtrsim Q$ holds if P and Q are bisimilar and, in addition, Q is at least as efficient as P, in the sense that Q is capable of producing the same activity as P without ever performing more internal activities—the τ-actions (we recall that $\stackrel{\mu}{\Longrightarrow}$ is $\Longrightarrow\stackrel{\mu}{\longrightarrow}\Longrightarrow$ even if $\mu = \tau$):

Definition 5.1.17 (Expansion progression)**.** We write $\mathcal{R} \stackrel{e}{\rightarrowtail} \mathcal{S}$ when $P\ \mathcal{R}\ Q$ implies:

- if $P \stackrel{\mu}{\longrightarrow} P'$ then $Q \stackrel{\hat{\mu}}{\longrightarrow} Q'$ and $P'\ \mathcal{S}\ Q'$.
- if $Q \stackrel{\mu}{\longrightarrow} Q'$ then $P \stackrel{\mu}{\Longrightarrow} P'$ and $P'\ \mathcal{S}\ Q'$;

We say $\mathcal{R}$ is an *expansion relation* when $\mathcal{R} \stackrel{e}{\rightarrowtail} \mathcal{R}$ and write $\gtrsim$ for the largest expansion relation, and simply call it *expansion*. Alternatively, $\gtrsim = \mathsf{gfp}(\mathbf{e})$ where $\mathbf{e} : \mathcal{S} \mapsto \mathcal{S} \cap \bigcup_{\mathcal{R} \stackrel{e}{\rightarrowtail} \mathcal{S}} \mathcal{R}$.

Up to expansion yields a function ($\mathcal{R} \mapsto \gtrsim\mathcal{R}\lesssim$) that is $\mathbf{wb}$-compatible. As a consequence, the same holds for the 'up to expansion and contexts' function. In fact, we care less about compatibility properties for $\mathbf{e}$ than those for $\mathbf{b}$ and $\mathbf{wb}$, as $\gtrsim$ is mainly used as a technique for $\approx$. (Though we will sometimes consider $\mathbf{e}$-compatible functions, for example $\mathcal{R} \mapsto \gtrsim\mathcal{R}\sim$.)

More sophisticated up-to techniques can be obtained by carefully adjusting the interplay between visible and internal transitions, and by taking into account termination hypotheses [Pou08, PS12].

Some further compatible functions are the functions $\mathbf{b}$ and $\mathbf{wb}$ themselves (indeed a function f is b-compatible if $f \circ b \subseteq b \circ f$, hence trivially f can be replaced by b itself). Intuitively, the use of $\mathbf{b}$ and $\mathbf{wb}$ as up-to techniques means that, in a diagram-chasing argument, the two derivatives need not be related; it is sufficient that the derivatives of such derivatives be related. Accordingly, we sometimes call functions $\mathbf{b}$ and $\mathbf{wb}$ *unfolding* functions. We will use $\mathbf{b}$ in the example in Section 5.4.1 and $\mathbf{wb}$ in Sections 5.3 and 5.4, when proving the $\mathbf{wb}$-compatibility of the up to context techniques.

Terminology We will simply say that a function is *compatible* to mean that it is both $\mathbf{b}$-compatible and $\mathbf{wb}$-compatible; similarly for compatibility up to.

5.1.1 Initial contexts

In this chapter, we establish LTSes for which bisimilarities ($\mathbf{b}$- and/or $\mathbf{wb}$-similarities) capture the equivalences of interest. We inherit all the theory of up-to techniques, along with the functions already proved ($\mathbf{b}$- or $\mathbf{wb}$-) compatible. These functions, although useful, are not specific to each LTS, and hence we want to establish the compatibility of the techniques specific to the syntax the corresponding calculus. Hence, we consider the "up to context" techniques, which are particularly useful. Those techniques can typically be decomposed into smaller functions.

In languages defined from a grammar, an n-ary context C is a term with numbered holes $[\cdot]_1$, ..., $[\cdot]_n$, and each hole $[\cdot]_i$ can appear any number of times in C. When $\widetilde{P}$ is a sequence of n processes $P_1, \ldots, P_n$, we write $C[\widetilde{P}]$ for C where each $[\cdot]_i$ is replaced with P_i. Using the same notation for $\widetilde{Q}$, we write $\widetilde{P}\ \mathcal{R}\ \widetilde{Q}$ when $\forall i\ P_i\ \mathcal{R}\ Q_i$.

We recall here the example of CCS [PS12], which is a particularly nice example where the up to context function can be decomposed using one function per operator of the calculus. The up to context function is the context closure function $\mathcal{C}$, defined as follows:

$$\mathcal{C} : \mathcal{R} \mapsto \{(C[\widetilde{P}], C[\widetilde{Q}]) \;\middle|\; C \text{ is a CCS context and } \widetilde{P}\ \mathcal{R}\ \widetilde{Q}\} \ .$$

We will use in the following a different—inductive—presentation of the same definition:

$$\frac{\widetilde{P}\ \mathcal{R}\ \widetilde{Q} \qquad C \text{ is a CCS context}}{C[\widetilde{P}]\ \mathcal{C}(\mathcal{R})\ C[\widetilde{Q}]} \ .$$

Since arities are finite, defining functions this way makes it obvious that they are continuous, in light of Remark 5.1.12.

To prove the compatibility of $\mathcal{C}$ directly, one needs to analyse the transitions of $C[\widetilde{P}]$, which usually involves an induction on C, and some amount of proof redundancy. Another way is to say that contexts can be decomposed into the following *initial contexts*, each being a polyadic context corresponding to one operator, or just simply a hole:

$$[\cdot]_1 \qquad 0 \qquad [\cdot]_1 \mid [\cdot]_2 \qquad (\boldsymbol{\nu} a)[\cdot]_1 \qquad \overline{a}.[\cdot]_1 \qquad a.[\cdot]_1 \qquad ![\cdot]_1 \qquad [\cdot]_1 + [\cdot]_2$$

Each initial context induces a function on relations. For example, $[\cdot]_1$ induces $\mathsf{id} : \mathcal{R} \mapsto \mathcal{R}$. We explicit the seven other functions below, one for each form of context:

$$\frac{}{0\ \ \mathcal{C}_0(\mathcal{R})\ \ 0} \qquad \frac{P\ \ \mathcal{R}\ \ Q \qquad a \in \mathcal{N}}{(\boldsymbol{\nu} a)P\ \ \mathcal{C}_\nu(\mathcal{R})\ \ (\boldsymbol{\nu} a)Q} \qquad \frac{P\ \ \mathcal{R}\ \ Q \qquad a \in \mathcal{N}}{\overline{a}.P\ \ \mathcal{C}_o(\mathcal{R})\ \ \overline{a}.Q} \qquad \frac{P\ \ \mathcal{R}\ \ Q \qquad a \in \mathcal{N}}{a.P\ \ \mathcal{C}_i(\mathcal{R})\ \ a.Q}$$

$$\frac{P\ \ \mathcal{R}\ \ Q}{!P\ \ \mathcal{C}_!(\mathcal{R})\ \ !Q} \qquad \frac{P_1\ \ \mathcal{R}\ \ Q_1 \qquad P_2\ \ \mathcal{R}\ \ Q_2}{P_1 \mid Q_2\ \ \mathcal{C}_|(\mathcal{R})\ \ Q_1 \mid Q_2} \qquad \frac{P_1\ \ \mathcal{R}\ \ Q_1 \qquad P_2\ \ \mathcal{R}\ \ Q_2}{P_1 + Q_2\ \ \mathcal{C}_+(\mathcal{R})\ \ Q_1 + Q_2}$$

Let F be the set of these functions: $F \triangleq \{\mathsf{id}, \mathcal{C}_0, \mathcal{C}_\nu, \mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_!, \mathcal{C}_|, \mathcal{C}_+\}$. By structural induction on the context C, we can prove that $\mathcal{C}$ is included in $(\cup F)^\omega$, which means that $\mathcal{C} = (\cup F)^\omega$ (the other inclusion holds since $\mathcal{C}$ is idempotent). It is possible to establish the $\mathbf{b}$-compatibility of $\mathcal{C}$ by proving the following evolutions:

$$\mathcal{C}_0 \overset{\mathbf{b}}{\rightsquigarrow} \mathsf{id} \qquad \mathcal{C}_\nu \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_\nu \qquad \mathcal{C}_o \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_o \cup \mathsf{id} \qquad \mathcal{C}_i \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_i \cup \mathsf{id} \qquad \mathcal{C}_| \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_| \qquad \mathcal{C}_+ \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_+ \cup \mathsf{id} \qquad \mathcal{C}_! \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_|^\omega \circ (\mathcal{C}_! \cup \mathsf{id})$$

and then invoking Lemma 5.1.11. In addition, each of these functions is $\mathbf{wb}$-compatible up to, except $\mathcal{C}_+$ (since $\approx$ is not preserved by sum contexts, for the reasons explained in Section 2.1.5).

5.2 The π-calculi

We define here a translation of the π-calculus LTS into a first-order LTS that follows the schema of abstract machines for the π-calculus (e.g., [Tur96]) in which the issue of the choice of fresh names is resolved by ordering the names and indexing the processes with a name that represents an upper bound to the names occurring in the process. In this way, early bisimilarity is the easiest style of bisimilarity to capture, but we discuss also the other styles (ground, late, open).

5.2.1 Early bisimilarity

We consider the early transition system defined in Section 2.2. In the bound output label $\overline{a}(b)$, name b is a binder for the free occurrences of b in P' and, as such, it is subject to α-conversion. The definition of bisimilarity takes α-conversion into account. The clause for bound output of strong early bisimilarity (Definition 2.2.3) says that:

1. if $P \xrightarrow{\overline{a}(b)}_\pi P'$ and $b \notin \mathrm{fn}(Q)$ then $Q \xrightarrow{\overline{a}(b)}_\pi Q'$ for some Q' such that $P' \sim Q'$.

This means that the LTS is not first-order, as demanding syntactic equality between labels is not enough to ensure the freshness of b.

When translating the π-calculus semantics to a first-order one, α-conversion and the condition $b \notin \mathrm{fn}(Q)$ have to be removed. To this end, one has to force an agreement between two bisimilar processes on the choice of the bound names appearing in transitions. We obtain this by considering *named processes* (c, P) in which c is bigger than or equal to all names in P. For this to make sense we assume an enumeration of the names and use $\leq$ as the underlying order, and $c+1$ for name following c in the enumeration; for a set of names N, we also write $c \geq N$ to mean $c \geq a$ for all $a \in N$. The state space *Pr* is now instantiated with the set named processes:

$$\mathit{Pr} = \{(c, P) \mid \mathrm{fn}(P) \leq c\}$$

The rules in Figure 5.3 define the translation of the π-calculus transition system to a first-order LTS. In the first-order LTS, the grammar for labels (*Act*) is the same as that of the original LTS:

$$\mathit{Act} : \mu \quad ::= \quad \tau \;\mid\; ab \;\mid\; \overline{a}b \;\mid\; \overline{a}(b)$$

However, for a named process (c, P) the only name that may be exported in a bound output is $c+1$; similarly only names that are below or equal to $c+1$ may be imported in an input transition. (Indeed, testing for all fresh names $b > c$ is unnecessary, doing it only for one ($b = c+1$) is enough.) This makes it possible to use the ordinary definition of bisimilarity for first-order LTS, and thus recover the early bisimilarity on the source terms.

Each rule of Figure 5.3 has, as its main premise, a $\dot{\longrightarrow}_\pi$-transition, using the LTS for π defined in Figure 2.1. Note that one could also build directly an equivalent LTS directly on named processes without using $\dot{\longrightarrow}_\pi$, but the correspondence is more clear this way.

This approach also allows us to consider extensions of π that can be described with the same kind of transitions. In fact, in the following, we will handle the π-calculus with choice and replication (Figure 2.1 augmented with the rules from (2.6) and (2.7) (Section 2.1.5)).

$$\frac{P \xrightarrow{\tau}_\pi P'}{(c,P) \xrightarrow{\tau} (c,P')} \qquad \frac{P \xrightarrow{ab}_\pi P'}{(c,P) \xrightarrow{ab} (c,P')}\; b \leq c \qquad \frac{P \xrightarrow{\overline{a}b}_\pi P'}{(c,P) \xrightarrow{\overline{a}b} (c,P')}\; b \leq c$$

$$\frac{P \xrightarrow{ab}_\pi P'}{(c,P) \xrightarrow{ab} (b,P')}\; b = c+1 \qquad \frac{P \xrightarrow{\overline{a}(b)}_\pi P'}{(c,P) \xrightarrow{\overline{a}(b)} (b,P')}\; b = c+1$$

Figure 5.3: First-order LTS for early bisimilarity in named π-calculus processes

We write π^1 for the first-order LTS derived from the translation of the π-calculus in Figure 5.3. Although the labels of the source and target transitions have a similar shape, the LTS in π^1 is first-order because labels are taken as purely syntactic objects (without α-conversion). We relate strong and weak early bisimilarities $\sim^{\mathrm{e}}$ and $\approx^{\mathrm{e}}$ to strong and weak standard bisimilarities in π^1:

Theorem 5.2.1. *Let P and Q be processes of π. Assume $c \geq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$. Then we have:*

- *$P \sim^{\mathrm{e}} Q$ iff $(c, P) \sim (c, Q)$, and*
- *$P \approx^{\mathrm{e}} Q$ iff $(c, P) \approx (c, Q)$.*

Proof. We prove the case of weak bisimilarity, the strong case being slightly easier. The following relation:

$$\mathcal{R}_1 \triangleq \{((c,P),(c,Q)) \;\mid\; P \approx^{\mathrm{e}} Q \;\wedge\; c \geq \mathrm{fn}(P) \cup \mathrm{fn}(Q)\}$$

is a weak bisimulation (in the sense of Definition 2.1.5). The only interesting transition is when $(c,P) \xrightarrow{\overline{a}(b)} (d,P')$, we know that $d = b = c+1$ and $P \xrightarrow{\overline{a}(b)}_\pi P'$. Since $c \geq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$, we know that $b \notin \mathrm{fn}(Q)$ so $P \approx^{\mathrm{e}} Q$ tells us that $Q \overset{\overline{a}(b)}{\Longrightarrow}_\pi Q'$ with $P' \approx^{\mathrm{e}} Q'$. Repeatedly applying rules from Figure 5.3, since $b = c+1$, we get:

$$\frac{Q \Longrightarrow_\pi Q_1 \xrightarrow{\overline{a}(b)}_\pi Q_2 \Longrightarrow_\pi Q'}{(c,Q) \Longrightarrow (c,Q_1) \xrightarrow{\overline{a}(b)} (b,Q_2) \Longrightarrow (b,Q')}$$

and indeed, $b \geq \mathrm{fn}(P') \cup \mathrm{fn}(Q')$, and $\mathcal{R}_1$ is a weak bisimulation.

For the converse, proving that

$$\mathcal{R}_2 \triangleq \{(P,Q) \;\mid\; \exists c \geq \mathrm{fn}(P) \cup \mathrm{fn}(Q) \;\; (c,P) \approx (c,Q)\}$$

is a weak early bisimulation (Definition 2.2.4) needs a little more care, are there are more labels to handle. Suppose $P \;\mathcal{R}_2\; Q$, which means there is a $c \geq \mathrm{fn}(P) \cup \mathrm{fn}(Q)$ such that $(c,P) \approx (c,Q)$. We analyse the transitions of the form $P \xrightarrow{\mu}_\pi P'$:

1. if $\mu = \overline{a}(b)$ or $\mu = ab$ and $b \geq c+2$ then we have $P \xrightarrow{\mu}_\pi P'$ if and only $P \overset{\mu\{b'/b\}}{\longrightarrow}_\pi P'\{b'/b\}$ with $b' = c+1$, and similarly for $Q \overset{\mu}{\Longrightarrow}_\pi Q'$, so we need only to handle the cases for which $\mathrm{n}(\mu) \leq c+1$.

2. if $\mu = ab$ or $\mu = \overline{a}(b)$ with $b = c+1$ then $P \xrightarrow{\mu}_\pi P'$ implies $(c,P) \xrightarrow{\mu} (b,P')$ which implies $(c,Q) \overset{\mu}{\Longrightarrow} (b,Q')$ and then $Q \overset{\mu}{\Longrightarrow}_\pi Q'$ with $(b,P') \approx (b,Q')$.

3. If $\mathrm{n}(\mu) \leq c$, there are two cases:

   (a) μ is not a bound output. Then $P \xrightarrow{\mu}_\pi P'$ if and only if $(c,P) \xrightarrow{\mu} (c,P')$ when $c \geq \mathrm{fn}(P)$, so getting $Q \overset{\mu}{\Longrightarrow}_\pi Q'$ and $P' \;\mathcal{R}_2\; Q'$ is straightforward.

   (b) $\mu = \overline{a}(b)$ (hence $b \notin \mathrm{fn}(P)$) with the additional information that $b \notin \mathrm{fn}(Q)$. Then $P \xrightarrow{\overline{a}(b')}_\pi P'\{b'/b\}$ with $b' = c+1$. We get $(c,P) \xrightarrow{\overline{a}(b')} (b', P'\{b'/b\})$ and then $(c,Q) \overset{\overline{a}(b')}{\Longrightarrow} (b',Q')$ and since $b \notin \mathrm{fn}(Q)$, we also have $Q \overset{\overline{a}(b)}{\Longrightarrow}_\pi Q'\{b/b'\}$. The progression also gives us $(b', P'\{b'/b\}) \approx (b',Q')$, on which we apply Lemmas 5.1.9 and 5.2.2 to obtain $(b,(P'\{b'/b\})\{b/b'\}) = (b,P') \approx (b,Q'\{b/b'\})$ and finally $P' \;\mathcal{R}_2\; Q'\{b/b'\}$.

(We used Lemma 5.2.2, proved in the following.) □

The above full abstraction result allows us to import the theory of up-to techniques for first-order LTSes and bisimilarity, both in the strong and the weak case. We have however to prove the soundness of up-to techniques that are specific to the π-calculus. These include two up-to techniques that allow us to apply injective substitutions to the names, and the up to context technique. Function isub implements 'up to injective name substitutions':

$$\mathsf{isub}(\mathcal{R}) \triangleq \{((d,P\sigma),(d,Q\sigma)) \;\; \text{s.t.} \;\; \begin{array}{l}(c,P)\;\mathcal{R}\;(c,Q),\\ \mathrm{fn}(P\sigma) \cup \mathrm{fn}(Q\sigma) \leq d, \text{ and}\\ \sigma \text{ is injective on } \mathrm{fn}(P) \cup \mathrm{fn}(Q)\;\}\;.\end{array}$$

A subtle drawback is the need of another function manipulating names, str, allowing us to replace the index c in a named process (c,P) with a lower one:

$$\mathsf{str}(\mathcal{R}) \triangleq \{((d,P),(d,Q)) \;\; \text{s.t. } (c,P)\;\mathcal{R}\;(c,Q) \text{ and } \mathrm{fn}(P,Q) \leq d\;\}\;.$$

Lemma 5.2.2. *The set* $\{\mathsf{isub}, \mathsf{str}\}$ *is compatible up to.*

Proof. We decompose isub into two functions: $\mathsf{isub} = \mathsf{bsub} \circ w$ where

1. $\mathsf{bsub} : \mathcal{R} \mapsto \{((n, P\sigma), (n, Q\sigma)) \mid (n, P)\ \mathcal{R}\ (n, Q)\ ,\ \sigma : n \to n \text{ is bijective}\}$ (up to bijective substitutions)
2. $w : \mathcal{R} \mapsto \{((n+k, P), (n+k, Q)) \mid (n, P)\ \mathcal{R}\ (n, Q)\ ,\ k \in \mathbb{N}\}$.

For both $b = \mathbf{b}$ and $b = \mathbf{wb}$, we prove that $\mathsf{bsub} \overset{b}{\rightsquigarrow} \mathsf{bsub}$, and that $w \overset{b}{\rightsquigarrow} \mathsf{bsub} \circ w$. From that $\mathsf{isub} \overset{b}{\rightsquigarrow} \mathsf{bsub} \circ \mathsf{isub} = \mathsf{isub}$. The same way, $\mathsf{str} \overset{b}{\rightsquigarrow} \mathsf{str} \circ \mathsf{bsub}$. The iteration is idempotent, as the functions are Scott-continuous. □

The up to context function is decomposed into a set of smaller context functions, similarly to initial contexts, one for each operator of the π-calculus. The only exception to this is the input prefix, since early bisimilarity in the π-calculus is not preserved by this operator. We write $\mathcal{C}_0$, $\mathcal{C}_o$, $\mathcal{C}_\nu$, $\mathcal{C}_!$, $\mathcal{C}_|$, and $\mathcal{C}_+$ for these initial context functions, respectively returning the closure of a relation under the operators of nil, output prefix, restriction, replication, parallel composition, and sum.

Definition 5.2.3. We define the functions $\mathcal{C}_0$, $\mathcal{C}_o$, $\mathcal{C}_\nu$, $\mathcal{C}_!$, $\mathcal{C}_|$ and $\mathcal{C}_+$ on relations on π^1 by the following rules:

$$\frac{}{(c,0)\ \ \mathcal{C}_0(\mathcal{R})\ \ (c,0)} \qquad \frac{(c,P)\ \mathcal{R}\ (c,Q) \qquad c \geq a,b \in \mathcal{N}}{(c,\overline{a}b.P)\ \ \mathcal{C}_o(\mathcal{R})\ \ (c,\overline{a}b.Q)} \qquad \frac{(c,P)\ \mathcal{R}\ (c,Q) \qquad a \in \mathcal{N}}{(c,(\boldsymbol{\nu} a)P)\ \ \mathcal{C}_\nu(\mathcal{R})\ \ (c,(\boldsymbol{\nu} a)Q)}$$

$$\frac{(c,P)\ \mathcal{R}\ (c,Q)}{(c,!P)\ \ \mathcal{C}_!(\mathcal{R})\ \ (c,!Q)} \qquad \frac{(c,P_1)\ \mathcal{R}\ (c,Q_1) \qquad (c,P_2)\ \mathcal{R}\ (c,Q_2)}{(c,P_1 \mid P_2)\ \ \mathcal{C}_|(\mathcal{R})\ \ (c,Q_1 \mid Q_2)} \qquad \frac{(c,P_1)\ \mathcal{R}\ (c,Q_1) \qquad (c,P_2)\ \mathcal{R}\ (c,Q_2)}{(c,P_1+P_2)\ \ \mathcal{C}_+(\mathcal{R})\ \ (c,Q_1+Q_2)}$$

Bisimilarity in the π-calculus is not preserved by input prefix. However, a weaker implication, requiring that the processes are bisimilar for every instantiation of the object, holds: (where $=$ can be $\sim^{\mathrm{e}}$ or $\approx^{\mathrm{e}}$)

$$\frac{P = Q \quad \text{and } P\{c/b\} = Q\{c/b\} \text{ for each } c \text{ free in } P, Q}{a(b).P = a(b).Q} \tag{5.4}$$

Definition 5.2.4. We define $\mathcal{C}_i$, the function for input prefixes, accordingly:

$$\frac{a \leq d \qquad \text{for all } c \leq d+1 \text{ we have } (d+1, P\{c/b\})\ \mathcal{R}\ (d+1, Q\{c/b\})}{(d, a(b).P)\ \ \mathcal{C}_i(\mathcal{R})\ \ (d, a(b).Q)}$$

Theorem 5.2.5. *The set* $\{\mathcal{C}_0, \mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_\nu, \mathcal{C}_!, \mathcal{C}_|, \mathcal{C}_+\}$ *is* $\mathbf{b}$*-compatible up to* $\mathsf{isub} \cup \mathsf{str}$.

Proof. We establish an evolution for each function. The proofs largely follow the ones from [Pou08] (and corresponding to Section 5.1.1). We account for the differences, which arise when name-passing is involved.

For each context function f, we prove $f \overset{\mathbf{b}}{\rightsquigarrow} g$ for some g combination of context functions and of $\mathsf{isub} \cup \mathsf{str}$. For each evolution, we assume $\mathcal{R} \rightarrowtail \mathcal{S}$ and $\mathcal{R} \subseteq \mathcal{S}$, and we prove $f(\mathcal{R}) \rightarrowtail g(\mathcal{S})$ (and we check that $f(\mathcal{R}) \subseteq g(\mathcal{S})$). The following evolutions are trivial:

$$\mathcal{C}_0 \overset{\mathbf{b}}{\rightsquigarrow} \mathsf{id} \qquad \mathcal{C}_o \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_o \cup \mathsf{id} \qquad \mathcal{C}_+ \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_+ \cup \mathsf{id}\ .$$

We give more details for the following evolutions. When proving $f(\mathcal{R}) \rightarrowtail g(\mathcal{S})$, we analyse transitions from the left-hand side, as the functions are all symmetric.

- $\mathcal{C}_i \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_i \cup \mathsf{str}$
  Assume $(d, a(b).P)\ \ \mathcal{C}_i(\mathcal{R})\ \ (d, a(b).Q)$. Each transition is of the form $\xrightarrow{ac}$, yielding a pair (p, q) where $p = (d', P\{c/b\})$ and $q = (d', Q\{c/b\})$.
  - if $d' = c+1$ then $(p, q) \in \mathcal{R}$ by definition of $\mathcal{C}_i$.
  - if $d' = c$ then $(d+1, P\{c/b\})\ \mathcal{R}\ (d+1, Q\{c/b\})$ by definition of $\mathcal{C}_i$, and hence $(p, q) \in \mathsf{str}(\mathcal{R})$.

- $\mathcal{C}_\nu \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_\nu \cup \mathsf{isub}$
  The interesting case arises for transitions for which the last rule applied is the extrusion rule: $(c, (\boldsymbol{\nu} d)P) \xrightarrow{\overline{a}(b)} (b, P'\{b/d\})$ with $b = c + 1$ and $P \xrightarrow{\overline{a}d}_\pi P'$. The problem is to relate $(b, P'\{b/d\})$ to $(b, Q'\{b/d\})$ knowing that $(c, P')\ \mathcal{S}\ (c, Q')$, which is done using the isub function with the injective substitution $\{b/d\} : c \to b$.

- $\mathcal{C}_| \overset{\mathbf{b}}{\rightsquigarrow} N \circ \mathcal{C}_| \circ \mathsf{isub}$, where $N \triangleq (\mathsf{str} \circ \mathcal{C}_\nu) \cup \mathsf{id}$.
  We know that the transition involves only $(c, P_i) \xrightarrow{\mu_i} (b_i, P_i')$ with $b_i = c$, we progress to $\mathcal{C}_|(\mathcal{R} \cup \mathcal{S}) = \mathcal{C}_|(\mathcal{S})$, using the fact that $\mathsf{id} \subseteq N$ and $\mathsf{id} \subseteq \mathsf{isub}$ with a proof similar to the one of [Pou08]. Otherwise, if one or two of the b_i is $c + 1$, some work is needed. There are two cases:

  – The last rule is a parallel rule, from P_1 (P_2 being symmetric), with a label of the form $\overline{a}(b)$. We know $(c, P_1)\ \mathcal{R}\ (c, Q_1)$ and $(c, P_2)\ \mathcal{R}\ (c, Q_2)$ and, say:

  $$\frac{P_1 \xrightarrow{\overline{a}(b)}_\pi P_1'}{P_1 \mid P_2 \xrightarrow{\overline{a}(b)}_\pi P_1' \mid P_2}\ b \notin \mathrm{fn}(P_2)$$

  We have $b = c + 1$, following the rule for bound output. We get the following reductions in π^1, from (c, P_1), and (c, Q_1) using the progression $\mathcal{R} \rightarrowtail \mathcal{S}$:

  $$(c, P_1) \xrightarrow{\overline{a}(b)} (b, P_1') \qquad\qquad (c, Q_1) \xrightarrow{\overline{a}(b)} (b, Q_1')$$

  With $(b, P_1')\ \mathcal{S}\ (b, Q_1')$. We now need to relate the resulting processes:

  $$\frac{\dfrac{(b, P_1')\ \mathcal{S}\ (b, Q_1')}{(b, P_1')\ \mathsf{isub}(\mathcal{S})\ (b, Q_1')} \qquad \dfrac{\dfrac{(c, P_2)\ \mathcal{R}\ (c, Q_2)}{(c, P_2)\ \mathcal{S}\ (c, Q_2)}}{(b, P_2)\ \mathsf{isub}(\mathcal{S})\ (b, Q_2)}}{(b, P_1' \mid P_2)\ (\mathcal{C}_| \circ \mathsf{isub})(\mathcal{S})\ (b, Q_1' \mid Q_2)}\ .$$

  The same happens for the input transition $\xrightarrow{ab}$ when $b = c + 1$.

  – The last rule is a closing rule: we know $(c, P_1)\ \mathcal{R}\ (c, Q_1)$ and $(c, P_2)\ \mathcal{R}\ (c, Q_2)$ and, say:

  $$\frac{P_1 \xrightarrow{\overline{a}(b)}_\pi P_1' \qquad P_2 \xrightarrow{ab}_\pi P_2'}{P_1 \mid P_2 \xrightarrow{\tau}_\pi (\boldsymbol{\nu} b)(P_1' \mid P_2')}\ b \notin \mathrm{fn}(P_2)$$

  In fact, we can assume $b = c + 1$ as b is fresh on both sides. The two hypotheses can then be transformed into transitions in π_1:

  $$(c, P_1) \xrightarrow{\overline{a}(b)} (b, P_1') \qquad\qquad (c, P_2) \xrightarrow{ab} (b, P_2')\ .$$

  We have the same transition for Q_1 and Q_2. Using the hypothesis $\mathcal{R} \rightarrowtail \mathcal{S}$, we get named processes (b, Q_1') and (b, Q_2'), related through $\mathcal{S}$, which we can combine using the context functions:

  $$\frac{\dfrac{\dfrac{(b, P_1')\ \mathcal{S}\ (b, Q_1')}{(b, P_1' \mid P_2')} \quad \mathcal{C}_|(\mathcal{S}) \quad \dfrac{(b, P_2')\ \mathcal{S}\ (b, Q_2')}{(b, Q_2' \mid Q_1')}}{(b, (\boldsymbol{\nu} b)(P_1' \mid P_2'))} \quad \mathcal{C}_\nu(\mathcal{C}_|(\mathcal{S})) \quad (b, (\boldsymbol{\nu} b)(Q_2' \mid Q_1'))}{(c, (\boldsymbol{\nu} b)(P_1' \mid P_2'))\ \ \mathsf{str}(\mathcal{C}_\nu(\mathcal{C}_|(\mathcal{S})))\ \ (c, (\boldsymbol{\nu} b)(Q_2' \mid Q_1'))}\ .$$

  Finally $\mathcal{C}_|(\mathcal{R})$ progresses to $\mathsf{str}(\mathcal{C}_\nu(\mathcal{C}_|(\mathcal{S})))$ in the case of a closing transition. Note that for the communication rule, the progression goes to $\mathcal{C}_|(\mathcal{S})$. The same happens when only P_1 or P_2 is moving, using the fact that $\mathcal{R} \subseteq \mathcal{S}$.

- $\mathcal{C}_! \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_|^\omega \circ N \circ \mathcal{C}_|^\omega \circ (\mathcal{C}_! \cup \mathsf{id}) \circ \mathsf{isub}$

  We analyse the transition $(c, !P) \xrightarrow{\mu} (c', P')$. We adapt Lemma 5.17 from [Pou08] characterising the transitions from a replicated process, to the π-calculus: if $!P \xrightarrow{\mu}_\pi P'$ then one of the following holds:

  1. $P' = !P|P_0|P|\ldots|P$ with $P \xrightarrow{\mu}_\pi P_0$, or
  2. $\mu = \tau$ and $P' = !P|P_0|P|\ldots|P|P_1|P|\ldots|P$ with $P \xrightarrow{\overline{a}b}_\pi P_i$ and $P \xrightarrow{ab}_\pi P_{1-i}$, or
  3. $\mu = \tau$ and $P' = (\boldsymbol{\nu} b)(!P|P_0|P|\ldots|P|P_1)|P|\ldots|P$ with $P \xrightarrow{\overline{a}(b)}_\pi P_i$ and $P \xrightarrow{ab}_\pi P_{1-i}$.

  (The last case is unnecessary in the case of CCS.)
  In case 1 we have $c' \in \{c, c+1\}$, and $(c, P) \xrightarrow{\mu} (c', P_0)$ otherwise $c' = c$ (we write $b = c$).
  In case 2 we have $(c, P) \xrightarrow{\mu_i} (b, P_i)$ for each i, with $b = c$.
  In case 3 we have $(c, P) \xrightarrow{\mu_i} (b, P_i)$ for each i, with $b = c + 1$.
  In each case, we obtain, since $\mathcal{R} \rightarrowtail \mathcal{S}$ with $(c, P)\ \mathcal{R}\ (c, Q)$, a transition $(c, !Q) \xrightarrow{\mu} (c', Q')$ with Q' of the same shape, so we only need to relate (c', P') to (c', Q') knowing $(b, P_i)\ \mathcal{S}\ (b, Q_i)$. First, remark that $(b, P)\ \mathsf{isub}(\mathcal{S})\ (b, Q)$. We have now the following pairs in $f(\mathcal{S})$ where $f = (\mathcal{C}_! \cup \mathsf{id}) \circ \mathsf{isub}$:

  $$(b, !P)\ f(\mathcal{S})\ (b, !Q) \qquad (b, P_0)\ f(\mathcal{S})\ (b, Q_0) \qquad (b, P_1)\ f(\mathcal{S})\ (b, Q_1) \qquad (b, P)\ f(\mathcal{S})\ (b, Q)$$

  we apply now $\mathcal{C}_|$ several times to obtain the three pairs (with $\mathcal{S}_1 = \mathcal{C}_|^\omega(f(\mathcal{S}))$):

  $$\begin{array}{rcl} (b, !P|P_0|P|\ldots|P) & \mathcal{S}_1 & (b, !Q|Q_0|Q|\ldots|Q) \\ (b, !P|P_0|P|\ldots|P|P_1|P|\ldots|P) & \mathcal{S}_1 & (b, !Q|Q_0|Q|\ldots|Q|Q_1|P|\ldots|P) \\ (b, !P|P_0|P|\ldots|P|P_1) & \mathcal{S}_1 & (b, !Q|Q_0|Q|\ldots|Q|Q_1) \end{array}$$

  The first two pairs handle cases 1 and 2. For case 3 we need to apply $\mathcal{C}_\nu$ to add $(\boldsymbol{\nu} b)-$ and then str to get back from $(b, (\boldsymbol{\nu} b)-)$ to $(c, (\boldsymbol{\nu} b)-)$. We apply $\mathcal{C}_|^\omega$ again to add the missing $-|P|\ldots|P$ and we get (c, P') and (c, Q') in the relation $\mathcal{C}_|^\omega(\mathsf{str}(\mathcal{C}_\nu(\mathcal{S}_1)))$.

  We have proved the following progression:

  $$\mathcal{R} \rightarrowtail \mathcal{S}_1 \cup \mathcal{C}_|^\omega(\mathsf{str}(\mathcal{C}_\nu(\mathcal{S}_1))) \subseteq \mathcal{C}_|^\omega(N(\mathcal{S}_1))$$

  and hence, putting back together all the cases, the following evolution:

  $$\mathcal{C}_! \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_|^\omega \circ N \circ \mathcal{C}_|^\omega \circ (\mathcal{C}_! \cup \mathsf{id}) \circ \mathsf{isub}$$

  For $\mathcal{C}_|$ in CCS we had $\mathcal{C}_| \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_|$, and in π we had $\mathcal{C}_| \overset{\mathbf{b}}{\rightsquigarrow} N \circ \mathcal{C}_| \circ \mathsf{isub}$: one isub on the right to augment the set of names, and one N on the left to handle name extrusion. We have the same phenomenon for $\mathcal{C}_!$ (in CCS: $\mathcal{C}_! \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_|^\omega \circ (\mathcal{C}_! \cup \mathsf{id})$).

We have established evolutions from all functions in $F \triangleq \{\mathcal{C}_0, \mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_\nu, \mathcal{C}_!, \mathcal{C}_|, \mathcal{C}_+, \mathsf{isub}, \mathsf{str}\}$ to $((\cup F)^\omega)^2$ (there are two iterations in the evolution of $\mathcal{C}_!$). We conclude by establishing the idempotence of u^ω. Each of the functions in F is continuous in the sense of Remark 5.1.12 (note that $\mathcal{C}_i$ is continuous, too: $(c, P)\ \mathcal{C}_i(\mathcal{R})\ (c, Q)$ can only use a finite (bounded by $c + 1$) number of "requests" to $\mathcal{R}$), so the union is continuous as well and u^ω is idempotent. □

Weak bisimilarity is not preserved by sums, only by guarded sums, whose function is $\mathcal{C}_{g+} \triangleq \mathcal{C}_+^\omega \circ (\mathcal{C}_o \cup \mathcal{C}_i)$. Note that we use the 'up to bisimilarity' function $\mathcal{R} \mapsto \sim\mathcal{R}\sim$ which is **wb**-compatible.

Theorem 5.2.6. *The set* $\{\mathcal{C}_i, \mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_\nu, \mathcal{C}_!, \mathcal{C}_|, \mathcal{C}_{g+}\}$ *is* **wb***-compatible up to* $\mathsf{isub} \cup \mathsf{str} \cup (\mathcal{R} \mapsto \sim\mathcal{R}\sim)$.

Proof. The progressions are as in the proof of Theorem 5.2.5, except that general choice is not compatible in the weak case, and we need one more up-to technique for the case of the replication.

The proofs of the following progressions are similar the corresponding ones in the proof of Theorem 5.2.5, except for $\mathcal{C}_{g+}$ which is new but is treated as $\mathcal{C}_o$ and $\mathcal{C}_i$.

$$\mathcal{C}_0 \overset{\mathbf{wb}}{\rightsquigarrow} \mathsf{id} \qquad \mathcal{C}_o \overset{\mathbf{wb}}{\rightsquigarrow} \mathcal{C}_o \cup \mathsf{id} \qquad \mathcal{C}_i \overset{\mathbf{wb}}{\rightsquigarrow} \mathcal{C}_i \cup \mathsf{str} \qquad \mathcal{C}_\nu \overset{\mathbf{wb}}{\rightsquigarrow} \mathcal{C}_\nu \cup \mathsf{isub} \qquad \mathcal{C}_{g+} \overset{\mathbf{wb}}{\rightsquigarrow} \mathcal{C}_{g+} \cup \mathsf{str} \qquad \mathcal{C}_| \overset{\mathbf{wb}}{\rightsquigarrow} N \circ \mathcal{C}_| \ .$$

For the replication operator, only case 1 of the corresponding proof cannot be transported to the weak case. We have:

$$(c, !P) \xrightarrow{\tau} (c, !P \,|\, P_0 \,|\, P \,|\, \ldots \,|\, P) \qquad \text{with} \qquad (c, P) \xrightarrow{\tau} (c, P_0) \ .$$

We use the fact that $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$ to get from $(c, P)\ \mathcal{R}\ (c, Q)$ that $(c, Q) \xrightarrow{\tau}{}^{n} (c, Q_0)$. Then

- if $n > 0$ we have $(c, !Q) \Longrightarrow (c, !Q \,|\, Q_0 \,|\, \ldots \,|\, Q)$ and we conclude as before.
- if $n = 0$ then there is no transition from Q or $!Q$, we know $P_0\ \mathcal{S}\ Q$ but we cannot reach the desired form $(c, !Q \,|\, Q \,|\, \ldots \,|\, Q)$ with a transition. Instead, we remark that $(c, !Q) \sim (c, !Q \,|\, Q \,|\, \ldots \,|\, Q)$ and so we simply progress to the relation $\mathcal{S}_2 \sim$ where $\mathcal{S}_2 = \mathcal{C}_|^\omega(\mathcal{C}_!(\mathcal{S}) \cup \mathcal{S})$.

Compared to the strong case, we only need to compose (on the left) the right-hand side of the evolution with $\mathcal{R} \mapsto \sim\mathcal{R}\sim$. Note that all the functions are continuous ($\mathcal{R} \mapsto \sim\mathcal{R}\sim$ is even pointwise defined), so the omega-iteration of their union is idempotent as desired. □

The compatibility of these functions does not follow from results of up to context techniques in the π-calculus; instead, above, we prove them from scratch, with the benefit of having a separate proof for each initial context.

As a byproduct of the compatibility up to of these initial context functions, and using Lemma 5.1.8, we derive the standard substitutivity properties of strong and weak early bisimilarity, including the rule (5.4) for input prefix.

Corollary 5.2.7. *In the π-calculus, relations $\sim^{\mathrm{e}}$ and $\approx^{\mathrm{e}}$ are preserved by the operators of output prefix, replication, parallel composition, restriction; $\sim^{\mathrm{e}}$ is also preserved by sum, whereas $\approx^{\mathrm{e}}$ is only preserved by guarded sums. Moreover, rule (5.4) is valid both for $\sim^{\mathrm{e}}$ and $\approx^{\mathrm{e}}$.*

5.2.2 Other bisimilarities in name-passing calculi

Section 5.2.1 studies the early version of the bisimilarity in the π-calculus. We examine in this section how we can recover with this method various bisimilarities in the π-calculus and other name-passing calculi.

5.2.2.1 Ground bisimilarity

The notion of early bisimulation (Definition 2.2.3) can be weakened to ground bisimulation, where there is no possible aliasing when dealing with inputs, since we only need to test one (fresh) object:

Definition 5.2.8 (Ground bisimilarity in π)**.** A relation $\mathcal{R}$ on π-calculus processes is a strong ground bisimulation if, whenever $P\ \mathcal{R}\ Q$:

1. if $P \xrightarrow{\overline{a}(b)}_\pi P'$ and $b \notin \mathrm{fn}(Q)$ then $Q \xrightarrow{\overline{a}(b)}_\pi Q'$ for some Q' such that $P'\ \mathcal{R}\ Q'$,
2. if $P \xrightarrow{ab}_\pi P'$ and $b \notin \mathrm{fn}(Q)$ then $Q \xrightarrow{ab}_\pi Q'$ for some Q' such that $P'\ \mathcal{R}\ Q'$,
3. if $P \xrightarrow{\mu}_\pi P'$ and μ is either $\overline{a}b$ or τ, then $Q \xrightarrow{\mu}_\pi Q'$ for some Q' such that $P'\ \mathcal{R}\ Q'$,
4. the converse of (1), (2) and (3), on Q.

Ground bisimilarity, $\sim_{\mathrm{g}}$, is the union of all ground bisimulations.

Weak ground bisimulations and weak ground bisimilarity $\approx_{\mathrm{g}}$ are defined accordingly. We just remark that the input is now treated as a bound output, and can only involve fresh names. Looking at Figure 5.3, we only need to remove the rule for input when $b \leq c$, to obtain the LTS in Figure 5.4, which yields $\sim$ and $\approx$.

$$\frac{P \xrightarrow{\tau}_{\pi} P'}{(c,P) \xrightarrow{\tau} (c,P')} \qquad \frac{P \xrightarrow{\overline{a}b}_{\pi} P'}{(c,P) \xrightarrow{\overline{a}b} (c,P')}\ b \leq c$$

$$\frac{P \xrightarrow{ab}_{\pi} P'}{(c,P) \xrightarrow{ab} (b,P')}\ b = c+1 \qquad \frac{P \xrightarrow{\overline{a}(b)}_{\pi} P'}{(c,P) \xrightarrow{\overline{a}(b)} (b,P')}\ b = c+1$$

Figure 5.4: First-order LTS for ground bisimilarity in named π-calculus processes

Then (weak) bisimilarity corresponds to (weak) ground bisimilarity.

$$\sim = \sim_{\mathrm{g}} \qquad\qquad \approx = \approx_{\mathrm{g}}$$

A similar method can be used to tackle πI (Section 3.1).

5.2.2.2 Late bisimilarity

Late bisimilarity makes use of transitions $P \xrightarrow{a(b)}_{\pi} P'$ where b is bound, and the definition of bisimulation contains a quantification over names. To capture this bisimilarity in a first-order LTS we need to have two transitions for each input $a(b)$: one to fire the input a, leaving b uninstantiated, and another to instantiate b.

Configurations are defined as follows, where $\mathrm{fn}(P) \leq c$. The first is a regular configuration, and the second is called an abstraction:

$$\mathit{Pr} : C \quad ::= \quad (c,P) \ \mid\ (b)(c,P)$$

The usual label ab for an input is split into two labels $a\_$ and $\_b$, the grammar of labels is now:

$$\mathit{Act} : \mu \quad ::= \quad \tau \ \mid\ a\_ \ \mid\ \_b \ \mid\ \overline{a}b \ \mid\ \overline{a}(b)\ .$$

The rule for input is split into three rules: the first triggers the abstraction configuration, the second instantiates it with a new name (*à la* ground bisimilarity) which has been chosen beforehand (b itself), and the third aliases b with a name d below c.

$$\frac{P \xrightarrow{ab}_{\pi} P'}{(c,P) \xrightarrow{a\_} (b)(c,P')}\ b = c+1 \qquad \frac{}{(b)(c,P) \xrightarrow{\_b} (b,P)}\ b = c+1 \qquad \frac{}{(b)(c,P) \xrightarrow{\_d} (c,P\{d/b\})}\ d \leq c$$

This translation does yield full abstraction for both strong and weak late bisimilarities. However, the decomposition of an input transition into two steps prevents us from obtaining the compatibility of up to context techniques. Moreover, bisimulation candidates now contain two syntactical categories.

5.2.2.3 Open bisimilarity

Open bisimilarity [SW01] is a coinductively defined congruence in the π-calculus and does not need any closure on top of it (unlike like full bisimilarity ([SW01], Definition 2.4.13) which needs closure under substitution). This makes it a good candidate for corresponding to a first-order LTS. Open bisimulations make use of a notion of distinction, which specifies which names cannot be equated:

Definition 5.2.9. A *distinction* is a finite symmetric and irreflexive relation on names. A substitution σ *respects* a distinction D, written $D \vdash \sigma$, if $(x, y) \in D$ implies $\sigma(x) \neq \sigma(y)$.

Definition 5.2.10. $\{\sim_{\mathrm{o}}^{D} \mid D \text{ is a distinction}\}$ is the largest family of symmetric relations such that if $P \sim_{\mathrm{o}}^{D} Q$ and $D \vdash \sigma$, then

1. if $P\sigma \xrightarrow{\mu}_{\pi} P'$ and μ is not a bound output, then $Q\sigma \xrightarrow{\mu}_{\pi} \sim_{\mathrm{o}}^{D\sigma} P'$
2. if $P\sigma \xrightarrow{\overline{a}(b)}_{\pi} P'$ and $b \notin \mathrm{fn}(Q\sigma)$ then $Q\sigma \xrightarrow{\overline{a}(b)}_{\pi} \sim_{\mathrm{o}}^{D'}$ where $D' = D\sigma \cup (\{b\} \times \mathrm{fn}(P\sigma, Q\sigma))^{\leftrightarrow}$

(where $\mathcal{R}^{\leftrightarrow}$ is the symmetric closure of $\mathcal{R}$). Each $\sim_{\mathrm{o}}^{D}$ is called *open D-bisimilarity*.

To transform this into a first-order LTS, configurations must contain the current information of distinction D. Since the object of the bound output is still subject to a quantification, we need again the "c" of the named processes.

$$\mathit{Pr} = \{(D, c, P) \mid \mathrm{fn}(P) \leq c \text{ and } D \text{ is a distinction and } D \subseteq \{1 \dots c\}^2\}$$

Labels now contain a substitution σ, to specify which names are being fused during the transition:

$$\begin{array}{rcl} \mu_0 & ::= & \tau \mid ab \mid \overline{a}b \mid \overline{a}(b) \\ \mathit{Act} : \mu & ::= & (\sigma, \mu_0) \end{array}$$

Note that the silent transition τ would be (σ_0, τ) with σ_0 the identity substitution, but we do not consider here weak open bisimilarity.

In the following rules, we require that all substitutions are restricted to the set $\{1 \dots c\}$, i.e. $\sigma : c \to c$ (we could be more precise and say $\sigma : c \to d$ with $d \leq c$).

$$\frac{P\sigma \xrightarrow{\tau}_{\pi} P' \qquad D \vdash \sigma}{(D, c, P) \xrightarrow{\sigma,\tau} (D\sigma, c, P')} \qquad \frac{P\sigma \xrightarrow{ab}_{\pi} P' \qquad D \vdash \sigma}{(D, c, P) \xrightarrow{\sigma,ab} (D\sigma, c, P')}\ b \leq c \qquad \frac{P\sigma \xrightarrow{\overline{a}b}_{\pi} P' \qquad D \vdash \sigma}{(D, c, P) \xrightarrow{\sigma,\overline{a}b} (D\sigma, c, P')}\ b \leq c$$

$$\frac{P\sigma \xrightarrow{ab}_{\pi} P' \qquad D \vdash \sigma}{(D, c, P) \xrightarrow{\sigma,ab} (D\sigma, b, P')}\ b = c + 1 \qquad \frac{P\sigma \xrightarrow{\overline{a}(b)}_{\pi} P' \qquad D \vdash \sigma \qquad D' = D\sigma \cup (\{b\} \times \{1 \dots c\})}{(D, c, P) \xrightarrow{\sigma,\overline{a}(b)} (D', b, P')}\ b = c + 1$$

Each of the rules follows closely the clauses of the definition of open bisimulations, so we can derive the expected full abstraction:

$$P \sim_{\mathrm{o}}^{D} Q \qquad \text{iff} \qquad \exists c \geq \mathrm{fn}(P, Q)\ \ (D, c, P) \sim (D, c, Q)\ .$$

We do not go into proving the compatibility of the up to context technique, but since each transition of a compound process can be decomposed into one or zero transitions from each of its parts (contrarily to the late first-order labelled transition system, where two are sometimes needed), there is room for hope.

5.2.2.4 πP

A first-order LTS for πP is trivially derived from the clauses of its bisimulation (Definition 4.5.5), and is inspired from the one for the ground bisimilarity seen in Section 5.2.2.1 to handle the bound names in input and output transitions. Labels now contain plain conditions, to test entailment directly in the LTS. We also add a special label τ that is syntactically distinct from any $[\varphi]\tau$, to properly handle the weak behavioural theory.

$$\mathit{Act} : \mu \ ::= \ \varphi \mid \alpha(b) \mid \overline{\alpha}(b) \mid [\varphi]\tau \mid \tau$$

To ensure freshness of the bound objects, the state space consists of named πP processes:

$$Pr = \{(c, P) \mid P \in \pi\mathrm{P} \;\; \mathrm{fn}(P) \leq c\} \;\; .$$

The set of rules, one for each label, is defined in Figure 5.5. Note that we could require $\mathrm{n}(\varphi) \leq c$ when defining the transition for labels $[\varphi]\tau$ or φ to reduce the number of cases to check without changing the set of bisimulations. When considering strong bisimilarity, the last rule (for label τ) can be removed.

$$\frac{P \xrightarrow{\alpha(b)}_{\pi\mathrm{P}} P'}{(c,P) \xrightarrow{\alpha(b)} (b,P')}\; b = c+1 \qquad \frac{P \xrightarrow{\overline{\alpha}(b)}_{\pi\mathrm{P}} P'}{(c,P) \xrightarrow{\overline{\alpha}(b)} (b,P')}\; b = c+1$$

$$\frac{P \xrightarrow{[\varphi]\tau}_{\pi\mathrm{P}} P'}{(c,P) \xrightarrow{[\varphi]\tau} (c, P' \mid \varphi)}\; \mathrm{n}(\varphi) \leq c \qquad \frac{P \rhd \varphi}{(c,P) \xrightarrow{\varphi} (c,P)} \qquad \frac{P \xrightarrow{[c\curlyvee c]\tau}_{\pi\mathrm{P}} P'}{(c,P) \xrightarrow{\tau} (c,P')}$$

Figure 5.5: First-order LTS for bisimilarity in named πP processes

With this LTS, the induced notion of bisimilarity coincides with the one of Definition 4.5.5, and weak bisimilarity with the one of Definition 4.9.1.

The usual up to context techniques can be defined. We add $\mathcal{C}_/ : \mathcal{R} \mapsto \{((c,\varphi),(c,\varphi)) \mid \mathrm{n}(\varphi) \leq c\}$, and we adapt $\mathcal{C}_o$ and $\mathcal{C}_i$ to extended bound prefixes (note that $\mathcal{C}_i$ is simpler now), and incorporate the matching prefix $[\varphi]\tau$ with $\mathcal{C}_{[]\tau}$. There is no $\mathcal{C}_+$, but $\mathcal{C}_{g+}$, as general sum has no precise sense in πP. Functions isub and str, with similar proofs as in π, are even more useful in πP than in π.

The progressions are more involved and use the up to $\sim$ technique, since we need to sometimes move around and rewrite arc processes after a $[\varphi]\tau$ transition. The set of functions isub, str, $\mathcal{C}_0$, $\mathcal{C}_/$, $\mathcal{C}_{[]\tau}$, $\mathcal{C}_o$, $\mathcal{C}_i$, $\mathcal{C}_\nu$, $\mathcal{C}_!$, $\mathcal{C}_|$, $\mathcal{C}_+$ is then $\mathbf{b}$-compatible up to $f_\sim \triangleq \mathcal{R} \mapsto \sim\mathcal{R}\sim$:

$$\mathsf{isub} \overset{\mathbf{b}}{\rightsquigarrow} \mathsf{isub} \qquad \mathsf{str} \overset{\mathbf{b}}{\rightsquigarrow} \mathsf{str} \circ \mathsf{isub} \qquad \mathcal{C}_0 \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_0 \qquad \mathcal{C}_/ \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_/ \qquad \mathcal{C}_\nu \overset{\mathbf{b}}{\rightsquigarrow} f_\sim \circ \mathcal{C}_\nu \qquad \mathcal{C}_{[]\tau} \overset{\mathbf{b}}{\rightsquigarrow} \mathcal{C}_{[]\tau} \cup \mathcal{C}_| \circ (\mathcal{C}_/ \cup \mathsf{id})$$

$$f \overset{\mathbf{b}}{\rightsquigarrow} f \cup (\mathcal{C}_\nu \circ \mathcal{C}_| \circ (\mathcal{C}_/ \cup \mathsf{isub})) \text{ for } f \in \{\mathcal{C}_o, \mathcal{C}_i, \mathcal{C}_{g+}\} \qquad \mathcal{C}_| \overset{\mathbf{b}}{\rightsquigarrow} (\mathcal{C}_| \circ (\mathcal{C}_/ \cup \mathsf{id}) \circ \mathsf{str} \circ \mathcal{C}_\nu \circ \mathcal{C}_|) \cup (f_\sim \circ \mathcal{C}_|)$$

The proofs of these evolutions intensively use tools designed in Section 4.5, in particular Lemma 4.5.23 and analyses resembling the one in the proof of Lemma 4.5.34.

Note that each function evolves to a composition that involves itself, this is due to the $\xrightarrow{\varphi}$ transitions. Some functions need to move arcs around, so we need the $f_\sim$ function. Remark that $\mathcal{C}_\nu$ progresses to $\mathcal{C}_\nu$ combined with $f_\sim$, which is consistent with the proof of Lemma 4.5.32, where proving up to restriction needs the usage of the up to bisimilarity technique.

5.3 Call-by-name λ-calculus

To study the applicability of our approach to higher-order languages, we investigate the pure call-by-name λ-calculus, referred to as ΛN in the sequel.

We use M, N to range over the set Λ of λ-terms, and x, y, z to range over variables. The standard syntax of λ-terms, and the rules for call-by-name reduction, are recalled in Section 2.4. We assume the familiar concepts of free and bound variables and substitutions, and identify α-convertible terms. The only values are the λ-abstractions $\lambda x.M$. In this section and in the following one, results and definitions are presented on closed terms; extension to open terms is made using closing abstractions (i.e., abstracting on all free variables). The reduction relation of ΛN is $\longmapsto_{\mathsf{n}}$, and $\Longmapsto_{\mathsf{n}}$ is its reflexive and transitive closure.

As bisimilarity for the λ-calculus we consider *environmental bisimilarity* [SKS11, KLS11], which allows a set of up-to techniques richer than Abramsky's applicative bisimilarity [Abr89], even if the two notions actually coincide, together with contextual equivalence. Environmental bisimilarity makes a clear distinction between the tested terms and the environment. An element of an environmental bisimulation has, in addition to the tested terms M and N, a further component $\mathcal{E}$, the environment, which expresses the observer's current knowledge. When an input from the observer is required, the arguments supplied are terms that the observer can build using the current knowledge; that is, terms obtained by composing the values in $\mathcal{E}$ using the operators of the calculus. An *environmental relation* is a set of elements each of which is of the form $(\mathcal{E}, M, N)$ or $\mathcal{E}$, and where M, N are closed terms and $\mathcal{E}$ is a relation on closed values. We use $\mathcal{X}, \mathcal{Y}$ to range over environmental relations. In a triple $(\mathcal{E}, M, N)$ the relation component $\mathcal{E}$ is the *environment*, and M, N are the *tested terms*. We write $M\ \mathcal{X}_{\mathcal{E}}\ N$ for $(\mathcal{E}, M, N) \in \mathcal{X}$. We write $\mathcal{E}^\star$ for the closure of $\mathcal{E}$ under contexts. We only define the weak version of the bisimilarity; its strong version is obtained in the expected way.

Definition 5.3.1. An environmental relation $\mathcal{X}$ is an *environmental bisimulation* if

1. $M\ \mathcal{X}_{\mathcal{E}}\ N$ implies:
   (a) if $M \longmapsto_{\mathsf{n}} M'$ then $N \Longmapsto_{\mathsf{n}} N'$ and $M'\ \mathcal{X}_{\mathcal{E}}\ N'$;
   (b) if $M = V$ then $N \Longmapsto_{\mathsf{n}} W$ and $\mathcal{E} \cup \{(V, W)\} \in \mathcal{X}$ (V and W are values);
   (c) the converse of the above two conditions, on N;
2. if $\mathcal{E} \in \mathcal{X}$ then for all $(\lambda x.P, \lambda x.Q) \in \mathcal{E}$ and for all $(M, N) \in \mathcal{E}^\star$ it holds that $P\{M/x\}\ \mathcal{X}_{\mathcal{E}}\ Q\{N/x\}$.

Environmental bisimilarity, $\approx^{\mathsf{env}}$, is the largest environmental bisimulation.

For the translation of environmental bisimilarity to first-order, a few issues have to be resolved. For instance, an environmental bisimilarity contains both triples $(\mathcal{E}, M, N)$, and pure environments $\mathcal{E}$, which shows up in the difference between clauses (1) and (2) of Definition 5.3.1. Moreover, the input supplied to tested terms may be constructed using arbitrary contexts. This is an important difference, compared to the LTS for π.

We write ΛN^1 for the first-order LTS resulting from the translation of ΛN. The states of ΛN^1 are sequences of λ-terms in which only the last one need not be a value. We use Γ and Δ to range over sequences of values only; thus (Γ, M) indicates a sequence of λ-values followed by M; and Γ_i is the i-th element in Γ.

$$Pr = \{(\Gamma, M) \mid \Gamma \text{ sequence of closed values, } M \text{ a closed term}\}$$

For an environment $\mathcal{E}$, we write $\mathcal{E}_1$ for an ordered projection of the pairs in $\mathcal{E}$ on the first component, and $\mathcal{E}_2$ is the corresponding projection on the second component. In the translation, intuitively, a triple $(\mathcal{E}, M, N)$ of an environmental bisimulation is split into the two components $(\mathcal{E}_1, M)$ and $(\mathcal{E}_2, N)$. Similarly, an environment $\mathcal{E}$ is split into $\mathcal{E}_1$ and $\mathcal{E}_2$. We write $C[\Gamma]$ for the term obtained by replacing each hole $[\cdot]_i$ in C with the value Γ_i. The labels of ΛN^1 include the special label τ, and can also be of the form i, C where i is a integer and C a multi-hole context:

$$Act : \mu \quad ::= \quad \tau \ \mid\ i, C$$

The rules for transitions in ΛN^1 are as follows:

$$\frac{M \longmapsto_{\mathsf{n}} M'}{(\Gamma, M) \xrightarrow{\tau} (\Gamma, M')} \qquad \frac{\Gamma_i(C[\Gamma]) \longmapsto_{\mathsf{n}} M'}{\Gamma \xrightarrow{i,C} (\Gamma, M')} \tag{5.5}$$

The first rule says that if M reduces to M' in ΛN then M can also reduce in ΛN^1, in any environment. The second rule implements the observations in clause (2) of Definition 5.3.1: in an environment Γ (only containing values), the ith component of Γ (Γ_i) can be tested by supplying, as input, a term obtained by filling a context C with values from Γ itself (C must range over contexts of arity $|\Gamma|$). The label of the transition records the position i and the context chosen.

Theorem 5.3.2. *Suppose M and N are closed values and $\mathcal{E}$ is an environment. Then:*

- $M \approx^{env}_{\mathcal{E}} N$ *iff* $(\mathcal{E}_1, M) \approx (\mathcal{E}_2, N)$,
- $\mathcal{E} \in \approx^{env}$ *iff* $\mathcal{E}_1 \approx \mathcal{E}_2$.

Proof. **(⇒)** We show that if $\mathcal{X}$ is an environmental bisimulation then $\mathcal{X}^2$ is a (first-order) weak bisimulation, where $\mathcal{X}^2$ relates $(\mathcal{E}_1, M)$ to $(\mathcal{E}_2, N)$ when $(\mathcal{E}, M, N) \in \mathcal{X}$, and $\mathcal{E}_1$ to $\mathcal{E}_2$ when $\mathcal{E} \in \mathcal{X}$. By symmetry we consider only one direction: we suppose $x\ \mathcal{X}^2\ y$ and a transition $x \xrightarrow{\mu} x'$, and we build y' such that $y \stackrel{\hat{\mu}}{\Longrightarrow} y'$ and $x'\ \mathcal{X}^2\ y'$.

1. $\mu = \tau$: then $x = (\mathcal{E}_1, M) \xrightarrow{\tau} (\mathcal{E}_1, M') = x'$ with $M \longmapsto_{\mathsf{n}} M'$, and $y = (\mathcal{E}_2, N)$ with $M\ \mathcal{X}_{\mathcal{E}} N$. By definition of environmental bisimulation, $N \Longmapsto_{\mathsf{n}} N'$ with $M'\ \mathcal{X}_{\mathcal{E}} N'$ and hence $y \Longrightarrow y'$ with $y' = (\mathcal{E}_2, N')$ and $x'\ \mathcal{X}^2\ y'$.
2. $\mu = i, C$: then $(x, y) = (\mathcal{E}_1, \mathcal{E}_2)$ with $\mathcal{E} \in \mathcal{X}$, and $x' = (\mathcal{E}_1, P\{C[\mathcal{E}_1]/x\})$ with $\lambda x.P = (\mathcal{E}_1)_i$ and we chose $y' = (\mathcal{E}_2, Q\{C[\mathcal{E}_2]/x\})$ with $\lambda x.Q = (\mathcal{E}_2)_i$. Then by construction, $y \xrightarrow{i,C} y'$ and $x'\ \mathcal{X}^2\ y'$ because $(\lambda x.P, \lambda x.Q) \in \mathcal{E}$ and $(C[\mathcal{E}_1], C[\mathcal{E}_2]) \in \mathcal{E}^\star$.

(⇐) The correspondence is less direct, so instead of establishing a correspondence between weak bisimulations, we define the candidate relation on top of weak bisimilarity. The environmental relation $\mathcal{X}$ is defined as follows:

$$\mathcal{X} \triangleq \{(\Gamma \cdot \Delta, M, N) \mid (\Gamma, M) \approx (\Delta, N)\} \cup \{\Gamma \cdot \Delta \mid \Gamma \approx \Delta\}$$

(where Γ and Δ only contain values). We prove that $\mathcal{X}$ is an environmental bisimulation up to reduction [SKS11].

1. Suppose $M\ \mathcal{X}_{\Gamma\cdot\Delta}\ N$ (i.e. $(\Gamma, M) \approx (\Delta, N)$).
   (a) if $M \longmapsto_{\mathsf{n}} M'$ then $(\Gamma, M) \xrightarrow{\tau} (\Gamma, M')$ which implies $(\Delta, N) \Longrightarrow (\Delta, N')$ with $(\Gamma, M') \approx (\Delta, N')$ and hence $N \Longmapsto_{\mathsf{n}} N'$ with $M' \mathcal{X}_{\Gamma\cdot\Delta} N'$;
   (b) if $M = V$, we want that $N \Longmapsto_{\mathsf{n}} W$ and $\Gamma \cdot \Delta \cup \{(V, W)\} \in \mathcal{X}$.
   Since $(\Gamma, V) = x \xrightarrow{i,C} x'$ for some x', this implies that $(\Delta, N) \stackrel{i,C}{\Longrightarrow} y_3$ for some y_3, i.e. $(\Delta, N) = y_0 \Longrightarrow y_1 \xrightarrow{i,C} y_2 \Longrightarrow y_3$. Since y_1 has an i, C transition, y_1 is of the form (Δ, W) for some W. Since $y_0 \Longrightarrow y_1$ and $\xrightarrow{\tau}$ is deterministic, we get $y_0 \approx y_1$. By transitivity of $\approx$, we get $x \approx y_1$ and hence $(\Gamma, V) \approx (\Delta, W)$, and hence $(\Gamma, V) \cdot (\Delta, W) \in \mathcal{X}$.
   (c) the converse of the above two conditions, on N, holds, as $\approx$ is symmetric.
2. If $(\lambda x.P, \lambda x.Q) \in \Gamma \cdot \Delta \in \mathcal{X}$ and $(M, N) \in (\Gamma \cdot \Delta)^\star$, we prove that $P\{M/x\}\ \mathcal{X}_{\Gamma\cdot\Delta}\ N' \Longleftarrow Q\{N/x\}$ for some N', using the "environmental bisimulation up to reduction" technique.
   We just remark that $(\lambda x.P, \lambda x.Q) = (\Gamma_i, \Delta_i)$ for some i, and that $(M, N) = (C[\Gamma], C[\Delta])$ for some C. Then $\Gamma_i(C[\Gamma]) \longmapsto_{\mathsf{n}} P\{M/x\}$ so $\Gamma \xrightarrow{i,C} (\Gamma, P\{M/x\})$ must be answered with $\Delta \stackrel{i,C}{\Longrightarrow} (\Delta, N')$ (with $(\Gamma, P\{M/x\}) \approx (\Delta, N')$).
   The first transition of $\Delta \stackrel{i,C}{\Longrightarrow} (\Delta, N')$ is necessarily $\xrightarrow{i,C}$, and is deterministic, so it must come from the sequence of $\Delta_i(C[\Delta]) \longmapsto_{\mathsf{n}} Q\{N/x\} \Longmapsto_{\mathsf{n}} N'$ and $P\{M/x\}\ \mathcal{X}_{\Gamma\cdot\Delta}\ N' \Longleftarrow_{\mathsf{n}} Q\{N/x\}$.

Hence, $\mathcal{X}$ is a environmental bisimulation up to reduction, and $\mathcal{X} \subseteq \approx^{\mathsf{env}}$. □

(The theorem also holds for the strong versions of the bisimilarities.) Again, having established full abstraction with respect to a first-order transition system and ordinary bisimilarity, we can inherit the theory of bisimulation enhancements. We have however to check up-to techniques that are specific to environmental bisimilarity. A useful such technique is 'up to environment', which allows us to replace an environment with a

larger one; $\mathsf{w}(\mathcal{R})$ is the smallest relation that includes $\mathcal{R}$ and such that, whenever $(V,\Gamma,M)\ \mathsf{w}(\mathcal{R})\ (W,\Delta,N)$ then also $(\Gamma,M)\ \mathsf{w}(\mathcal{R})\ (\Delta,N)$, where V and W are values. In other terms:

$$\frac{(V_1,\ldots,V_n,\Gamma,M)\ \mathcal{R}\ (W_1,\ldots,W_n,\Delta,N)}{(\Gamma,M)\ \mathsf{w}(\mathcal{R})\ (\Delta,N)}$$

Here w stands for 'weakening' as, from Lemmas 5.1.8 and 5.3.3, if $(V,\Gamma,M)\approx(W,\Delta,N)$ then $(\Gamma,M)\approx(\Delta,N)$.

The technique "up to permutation" contains (Γ_π,Δ_π) (respectively $((\Gamma_\pi,M),(\Delta_\pi,N))$) when (Γ,Δ) (respectively $((\Gamma,M),(\Delta,N))$) is in the base relation, writing Γ_π for $(V_{\pi(1)},\ldots,V_{\pi(n)})$ when $\Gamma=(V_1,\ldots,V_n)$ and π is a permutation of $\{1,\ldots,n\}$. We use this technique without explicitly mentioning it, which is easily proven compatible and Scott-continuous.

Lemma 5.3.3. *Function* w *is compatible.*

Proof. Rather immediate: τ transitions do not alter the environment, and as for a i,C transition (writing $\Gamma'=V_1,\ldots,V_n,\Gamma$ and $\Delta'=W_1,\ldots,W_n,\Delta$)

$$\Gamma_i(C[\Gamma])=\Gamma'_{i+n}(C_{+n}[\Gamma]) \qquad\qquad \Delta_i(C[\Delta])=\Delta'_{i+n}(C_{+n}[\Delta])\ ,$$

where C_{+n} is the shifting of each hole $[\cdot]_j$ of C into $[\cdot]_{j+n}$. Then $\Gamma\xrightarrow{i,C}(\Gamma,M')$ implies $\Gamma'\xrightarrow{i+n,C_{+n}}(\Gamma',M')$ and $\Delta'\xrightarrow{i+n,C_{+n}}(\Delta',N')$ implies $\Delta\xrightarrow{i,C}(\Delta,N')$, and so from $\mathcal{R}\overset{w}{\rightarrowtail}\mathcal{S}$ we can build $\mathsf{w}(\mathcal{R})\overset{w}{\rightarrowtail}\mathsf{w}(\mathcal{S})$. Since w is monotone, $\mathcal{R}\subseteq\mathcal{S}$ implies $\mathsf{w}(\mathcal{R})\subseteq\mathsf{w}(\mathcal{S})$. □

Somehow dual to weakening is the strengthening of the environment, in which a component of an environment can be removed. However this is only possible if the component removed is 'redundant', that is, it can be obtained by gluing other pieces of the environment within a context; strengthening is captured by the following str function:

$$\frac{(\Gamma,M)\ \mathcal{R}\ (\Delta,N)\qquad C_v\text{ is a value context}}{(\Gamma,C_v[\Gamma],M)\ \mathsf{str}(\mathcal{R})\ (\Delta,C_v[\Delta],N)}$$

(C_v is a value context when the outermost operator of C_v is an abstraction). We derive the compatibility up to of str in Theorem 5.3.8.

For up to context, we need to distinguish between arbitrary contexts and evaluation contexts. There are indeed substitutivity properties, and corresponding up-to techniques, that only hold for the latter contexts. For example, with $\Omega\triangleq(\lambda x.xx)(\lambda x.xx)$, since we have $\Omega\longmapsto_{\mathsf{n}}\Omega$, we get $(\Gamma,\Omega)\sim(\Delta,\Omega)$ for all environments Γ and Δ. If we could apply substitutivity with context $C=(\lambda xy.y)[\cdot]$ to it, then we would obtain $(\Gamma,C[\Omega])\sim(\Delta,C[\Omega])$ which implies $(\Gamma,\lambda y.y)\sim(\Delta,\lambda y.y)$ (and then $\Gamma\sim\Delta$ using w), which is false for example with $\Gamma=(\lambda x.x)$ and $\Delta=(\lambda x.\Omega)$.

Soundness also does not hold for arbitrary contexts, for similar reasons: relation $\mathcal{R}\triangleq\{((\Gamma,\Omega),(\Delta,\Omega)),((\Gamma,(\lambda x.x)(\lambda y.\Omega)),(\Delta,(\lambda x.x)(\lambda y.\Omega)))\}$ is a bisimulation up to arbitrary contexts (the first pair progresses to itself, the second pair progresses to the first one using the context $\lambda y.[\cdot]$) but $\mathcal{R}\subseteq\sim$ would imply $(\Gamma,\lambda y.\Omega)\sim(\Delta,\lambda y.\Omega)$ and then $\Gamma\sim\Delta$ using w.

A hole $[\cdot]_i$ of a context C is in a *redex position* if the context obtained by filling all the holes but $[\cdot]_i$ with values is an evaluation context. We define below $\mathcal{C}$ ("up to arbitrary contexts, on values") and $\mathcal{C}_{\mathsf{e}}$ ("up to evaluation contexts") as follows:

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma,C[\Gamma])\ \mathcal{C}(\mathcal{R})\ (\Delta,C[\Delta])}\qquad\qquad\frac{(\Gamma,M)\ \mathcal{R}\ (\Delta,N)}{(\Gamma,E[M,\Gamma])\ \mathcal{C}_{\mathsf{e}}(\mathcal{R})\ (\Delta,E[N,\Delta])}\ .$$

Above, C ranges over arbitrary contexts, whereas E ranges over contexts whose first hole is in redex position. In the following, we prove that these functions are compatible up to.

First, we establish two technical lemmas that we will use several times in the compatibility proofs.

Lemma 5.3.4. *The functions* $\mathbf{b}'$*,* $\mathbf{e}'$ *and* $\mathbf{wb}'$ *defined below are respectively* $\mathbf{b}$*-,* $\mathbf{e}$*- and* $\mathbf{wb}$*-compatible:*

$$\mathbf{b}' : \mathcal{S} \mapsto \bigcup_{\mathcal{R} \rightarrowtail \mathcal{S}} \mathcal{R} \qquad \mathbf{e}' : \mathcal{S} \mapsto \bigcup_{\mathcal{R} \overset{e}{\rightarrowtail} \mathcal{S}} \mathcal{R} \qquad \mathbf{wb}' : \mathcal{S} \mapsto \bigcup_{\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}} \mathcal{R} \ .$$

Proof. First, remark that $\mathbf{b}'(\mathcal{S}_1 \cap \mathcal{S}_2) \subseteq \mathbf{b}'(\mathcal{S}_1) \cap \mathbf{b}'(\mathcal{S}_2)$, since when $\{(x,y)\} \rightarrowtail \mathcal{S}_1 \cap \mathcal{S}_2$ then $\{(x,y)\} \rightarrowtail \mathcal{S}_i$ for both i. Remark that $\mathbf{b} = \mathsf{id} \cap \mathbf{b}'$, we prove $\mathbf{b}' \circ \mathbf{b} \subseteq \mathbf{b} \circ \mathbf{b}'$. Let $\mathcal{S}$ be a relation. Because $\mathbf{b}'$ is monotone, $\mathbf{b}'(A \cap B) \subseteq \mathbf{b}'(A) \cap \mathbf{b}'(B)$, so we have:

$$\mathbf{b}'(\mathbf{b}(\mathcal{S})) = \mathbf{b}'(\mathcal{S} \cap \mathbf{b}'(\mathcal{S})) \subseteq \mathbf{b}'(\mathcal{S}) \cap \mathbf{b}'(\mathbf{b}'(\mathcal{S})) = \mathbf{b}(\mathbf{b}'(\mathcal{S})) \ .$$

We reason similarly for $\mathbf{e}'$ and $\mathbf{wb}'$. □

Note that $\mathbf{b}$, $\mathbf{b}'$, $\mathbf{e}$, $\mathbf{e}'$, $\mathbf{wb}$ and $\mathbf{wb}'$ are Scott-continuous when $\{(\mu, P') \mid P \xrightarrow{\mu} P'\}$ is finite for all P, even if $\Longrightarrow$ is not image-finite. Unfortunately, this is not the case for ΛN^1, so we will resort to Lemma 5.1.14 instead of Lemma 5.1.11 and have a more intricate witness. We use $\mathbf{wb}'$ in the following lemma:

Lemma 5.3.5. *In* ΛN^1 *(and in* ΛR^1 *of Section 5.4), if* $(\Gamma, y) \in \mathbf{wb}'(\mathcal{S})$ *then* $y \Longrightarrow \Delta$ *with* $(\Gamma, \Delta) \in \mathbf{wb}'(\gtrsim \mathcal{S} \lesssim)$.

Proof. We know that there is a label $\mu = 1, C_0$ (where C_0 is some context of arity $|\Gamma|$, for example the context $\lambda x.x$ with no hole) such that $\Gamma \xrightarrow{\mu} x_0$, that y must catch up with $y \overset{\mu}{\Longrightarrow} y'_0$ for some y'_0. Since μ is a transition that only configurations of values can make, there must be some Δ such that $y \Longrightarrow \Delta \xrightarrow{\mu} \Longrightarrow y'_0$. Note also that Γ and Δ have the same arities, and hence the same transitions.

We prove now $\{(\Gamma, \Delta)\} \overset{w}{\rightarrowtail} (\mathcal{S} \Longleftarrow)$. We analyse the transitions labelled with $\mu = i, C$ from both sides.

1. Suppose that $\Gamma \xrightarrow{\mu} x'$. Then $y \Longrightarrow y_1 \xrightarrow{\mu} y_2 \Longrightarrow y_3$ with $x' \ \mathcal{S} \ y_3$. Since y_1 must be the unique value Δ, we have $\Delta \overset{\mu}{\Longrightarrow} y_3$ as well.
2. Suppose that $\Delta \xrightarrow{\mu} y'$. Since Γ and Δ have the same transitions, we have $\Gamma \xrightarrow{\mu} x'$ for some other x'. Then we run the $\overset{w}{\rightarrowtail}$ progression *from left to right* to get $y \overset{\mu}{\Longrightarrow} y_3$ with $x' \ \mathcal{S} \ y_3$ again. Using the same notations as above, we have $y_1 = \Delta$ and $y_2 = y'$. Then we have $x' \ \mathcal{S} \Longleftarrow y'$.

In the end $\{(\Gamma, \Delta)\}$ weakly progresses to $\mathcal{S} \Longleftarrow$ which is included in $\gtrsim \mathcal{S} \lesssim$ since $\Longrightarrow \subseteq \gtrsim$ in a deterministic setting. □

Remark 5.3.6. Note that Lemma 5.3.5 is highly specific to the LTS: it works for ΛN^1 in this section and for ΛR^1 in the next, but the proof uses the fact that reduction is deterministic, and that values can have roughly the same sets of transitions. A more robust approach would be to have a label to declare that the configuration is composed of values:

$$\frac{}{\Gamma \xrightarrow{\lambda} \Gamma} \ .$$

This would get rid of the need of Lemma 5.3.5, and would tolerate nondeterminism. After all, "being a value" is the only observable in the equivalence we capture, so it is sensible to have the corresponding label.

Lemma 5.3.5 holds whenever the actions of the LTS are split into τ actions, for non-value configurations, and disjoints classes of visible actions, for value configurations, such that all the labels in each class are triggered at the same moment. In ΛN^1, this is exemplified by the following property: $\Gamma \xrightarrow{\mu}$ iff $\mu = i, C$ where $i \leq |\Gamma|$ and C is $|\Gamma|$-ary. This requires that the information of arity of the context is present in C (otherwise, we resort to adding the transition λ from Remark 5.3.6 to the LTS).

The following lemma holds, with a similar proof:

Lemma 5.3.7. *In* ΛN^1 *(and in* ΛR^1 *of Section 5.4) if* $(x, \Delta) \in \mathbf{e}'(\mathcal{S})$ *then* $x \Longrightarrow \Gamma$ *with* $(\Gamma, \Delta) \in \mathbf{e}'(\gtrsim \mathcal{S})$.

We will use Lemma 5.3.7 in Section 5.4.

Theorem 5.3.8. *The set* $\{\mathsf{str}, \mathcal{C}, \mathcal{C}_e\}$ *is:*

1. $\mathbf{b}$*-compatible up to the identity function.*
2. $\mathbf{wb}$*-compatible up to* $\mathbf{wb}' \cup f_{\gtrsim}$ *where* $f_{\gtrsim} \triangleq (\mathcal{R} \mapsto \gtrsim \mathcal{R} \lesssim)$ *is 'up to expansion' is proved compatible in [PS12].*

Proof. We only handle the weak case: the strong case is significantly easier, but hides important problems since the diagrams are symmetric. For each function f in $F = \{\mathsf{str}, \mathcal{C}, \mathcal{C}_e\}$ we build $g \subseteq (\cup F)^\omega$ such that $f \overset{\mathbf{wb}}{\rightsquigarrow} g$, in separate lemmas (Lemmas 5.3.9, 5.3.10 and 5.3.11). We establish these evolutions using diagram-chasing arguments: for each f in F we suppose $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$ and $\mathcal{R} \subseteq \mathcal{S}$ and we prove that $f(\mathcal{R}) \overset{w}{\rightarrowtail} g(\mathcal{S})$ (we also check that $f \subseteq g$). □

Lemma 5.3.9. $\mathsf{str} \overset{\mathbf{wb}}{\rightsquigarrow} \mathsf{str} \cup \mathsf{str} \circ \mathcal{C} \circ \mathbf{wb}' \circ f_{\gtrsim}$.

Proof. Let $(\Gamma, M)\ \mathcal{R}\ (\Delta, N)$. We study transitions from $(\Gamma, C_v[\Gamma], M)$, $(\Delta, C_v[\Delta], N)$.

1. if M is not a value then $(\Gamma, C_v[\Gamma], M) \xrightarrow{\tau} x'$ is the only transition. We know that $M \longmapsto_{\mathsf{n}} M'$ and x' is of the form $(\Gamma, C_v[\Gamma], M')$, and hence $(\Gamma, M) \xrightarrow{\tau} (\Gamma, M')$. Playing the progression game $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$, we get an N' such that $(\Delta, N) \Longrightarrow (\Delta, N')$ and hence $(\Delta, C_v[\Delta], , N) \Longrightarrow (\Delta, C_v[\Delta], N')$ with $(\Gamma, C_v[\Gamma], M')$ $\mathsf{str}(\mathcal{S})$ $(\Delta, C_v[\Delta], N')$.

$$\begin{array}{ccc} (\Gamma, M) & \mathcal{R} & (\Delta, N) \\ {\scriptstyle\tau}\downarrow & & \Downarrow \\ (\Gamma, M') & \mathcal{S} & (\Delta, N') \end{array} \quad \rightsquigarrow \quad \begin{array}{ccc} (\Gamma, C_v[\Gamma], M) & \mathsf{str}(\mathcal{R}) & (\Delta, C_v[\Delta], N) \\ {\scriptstyle\tau}\downarrow & & \Downarrow \\ (\Gamma, C_v[\Gamma], M') & \mathsf{str}(\mathcal{S}) & (\Delta, C_v[\Delta], N') \end{array}$$

2. If $M = V$ is a value, we consider the transition $\Gamma'' = (\Gamma, C_v[\Gamma], V) \xrightarrow{i, C} x' = (\Gamma'', M')$ which means $\Gamma''_i(C[\Gamma'']) \longmapsto_{\mathsf{n}} M'$. Depending on i, we handle the progression differently (we write $n \triangleq |\Gamma|$ and $\Gamma' \triangleq (\Gamma, V)$):

   (a) $i \leq n$ or $i = n + 2$: then we have $\Gamma''_i = \Gamma'_{i'}$ with $i' = \min(i, n + 1)$. We can also build a context C' from the composition of C with C_v such that $C[\Gamma''] = C'[\Gamma']$, yielding:

$$\begin{array}{ccc} \Gamma' & \mathcal{R} & (\Delta, N) \\ & & \Downarrow \\ {\scriptstyle i', C'}\downarrow & & \Delta' \\ & & \Downarrow i', C' \\ (\Gamma', M') & \mathcal{S} & (\Delta', N'_1) \end{array} \quad \rightsquigarrow \quad \begin{array}{ccc} \Gamma'' & \mathsf{str}(\mathcal{R}) & (\Delta, C_v[\Delta], N) \\ & & \Downarrow \\ {\scriptstyle i, C}\downarrow & & (\Delta, C_v[\Delta], W) \triangleq \Delta'' \\ & & \Downarrow i, C \\ (\Gamma'', M') & \mathsf{str}(\mathcal{S}) & (\Delta'', N'_1) \end{array}$$

   (Note that we use "up to permutation".)

   (b) $i = n + 1$. This time $\Gamma''_i = C_v[\Gamma]$, and hence $M' = C'[\Gamma']$ for some C', combining again C and C_v. We can apply Lemma 5.3.5 with $\Gamma' \triangleq (\Gamma, V)$ and (Δ, N) to get $(\Delta, N) \Longrightarrow \Delta' = (\Delta, W)$ for some W with $(\Gamma', \Delta') \in \mathbf{wb}'(\gtrsim \mathcal{S} \lesssim)$.

$$\begin{array}{ccccc}
\Gamma'' & \text{———} & \mathsf{str}(\mathcal{R}) & \text{———} & (\Delta, C_v[\Delta], N) \\
 & & & & \Downarrow \\
i, C \downarrow & & & & (\Delta, C_v[\Delta], W) \triangleq \Delta'' \\
 & & & & \Downarrow i, C \\
(\Gamma'', C'[\Gamma']) & \text{—} \mathsf{str}(\mathcal{C}(\mathbf{wb}'(f_{\gtrsim}(\mathcal{S})))) \text{—} & & & (\Delta'', C'[\Delta'])
\end{array}$$

Indeed, $\Gamma'\ \mathcal{S}_1\ \Delta'$ with $\mathcal{S}_1 \triangleq \mathbf{wb}'(f_{\gtrsim}(\mathcal{S}))$ so $(\Gamma', C'[\Gamma'])\ \mathcal{C}(\mathcal{S}_1)\ (\Delta', C'[\Delta'])$ and finally we get up to permutation: $(\Gamma'', C'[\Gamma'])\ \mathsf{str}(\mathcal{C}(\mathcal{S}_1))\ (\Delta'', C'[\Delta'])$.

□

Lemma 5.3.10. $\mathcal{C} \overset{\mathbf{wb}}{\rightsquigarrow} \mathsf{str} \cup \mathcal{C} \cup \mathcal{C}_\mathsf{e}$.

Proof. We know $\Gamma\ \mathcal{R}\ \Delta$ and we analyse transitions from $(\Gamma, C[\Gamma])$. If $C = [\cdot]_i$ then we progress to str. We suppose now $C \neq [\cdot]_i$. In the first two cases, we suppose that $C[\Gamma]$ is a value, hence C is a value context C_v.

1. $(\Gamma, C_v[\Gamma]) \xrightarrow{i,C} (\Gamma, C_v[\Gamma], M')$.
   (a) If $i = |\Gamma| + 1$ then M' is of the form $C'[\Gamma]$ and we end up in $\mathsf{str}(\mathcal{C}(\mathcal{R}))$.
   (b) If $i < |\Gamma| + 1$ then $\Gamma_i(C[\Gamma, C_v[\Gamma]]) \longmapsto_\mathsf{n} M'$ and since $C[\Gamma, C_v[\Gamma]] = C'[\Gamma]$ for some C' we get $\Gamma \xrightarrow{i,C'} (\Gamma, M')$, and similarly for Δ.

$$\begin{array}{ccc}
\Gamma & \text{——}\ \mathcal{R}\ \text{——} & \Delta \\
i, C' \downarrow & & \Downarrow i, C' \\
(\Gamma, M') & \text{—}\ \mathcal{S}\ \text{—} & (\Delta, N')
\end{array}
\quad \rightsquigarrow \quad
\begin{array}{ccccc}
(\Gamma, C_v[\Gamma]) & \text{——}\ \mathcal{C}(\mathcal{R})\ \text{——} & (\Delta, C_v[\Delta]) \\
i, C \downarrow & & \Downarrow i, C \\
(\Gamma, C_v[\Gamma], M') & \text{—}\ \mathsf{str}(\mathcal{S})\ \text{—} & (\Delta, C_v[\Delta], N')
\end{array}$$

   Note that this time, we moved from $\mathcal{C}(\mathcal{R})$ to $\mathsf{str}(\mathcal{S})$.

2. If $C[\Gamma]$ is not a value then $(\Gamma, C[\Gamma]) \xrightarrow{\tau} (\Gamma, M')$ with $C[\Gamma] \longmapsto_\mathsf{n} M'$. We distinguish two cases:
   (a) C has no hole in evaluation position. Then $M' = C'[\Gamma]$ for some C' and $C[\Delta] \longmapsto_\mathsf{n} C'[\Delta]$ and we end up in $\mathsf{str}(\mathcal{R})$.
   (b) $C[\Gamma] = E[\Gamma_i(C'[\Gamma]), \Gamma]$ where $\Gamma_i(C'[\Gamma])$ is in evaluation position, because Γ_i is in evaluation position and $C[\Gamma]$ is not a value (the same holds with Δ instead of Γ). Then $M' = E[M_1, \Gamma]$ with $\Gamma_i(C'[\Gamma]) \longmapsto_\mathsf{n} M_1$. Then we can do the same with $\Delta_i(C'[\Delta])$ using a i, C' transition, and using $\mathcal{C}_\mathsf{e}$:

$$\begin{array}{ccc}
\Gamma & \text{——}\ \mathcal{R}\ \text{——} & \Delta \\
i, C' \downarrow & & \Downarrow i, C' \\
(\Gamma, M_1) & \text{—}\ \mathcal{S}\ \text{—} & (\Delta, N_1)
\end{array}
\quad \rightsquigarrow \quad
\begin{array}{ccccc}
(\Gamma, E[\Gamma_i(C'[\Gamma]), \Gamma]) & \text{—}\ \mathcal{C}(\mathcal{R})\ \text{—} & (\Delta, E[\Delta_i(C'[\Delta]), \Delta]) \\
\tau \downarrow & & \Downarrow \tau \\
(\Gamma, E[M_1, \Gamma]) & \text{——}\ \mathcal{C}_\mathsf{e}(\mathcal{S})\ \text{——} & (\Delta, E[N_1, \Delta])
\end{array}$$

□

Lemma 5.3.11. $\mathcal{C}_\mathsf{e} \overset{\mathbf{wb}}{\rightsquigarrow} \mathsf{w} \circ (\mathcal{C} \cup \mathcal{C}_\mathsf{e} \cup \mathsf{str} \cup \mathsf{str} \circ \mathcal{C}) \circ (\mathbf{wb}' \circ f_{\gtrsim} \cup \mathsf{id})$.

Proof. We know $(\Gamma, M)\ \mathcal{R}\ (\Delta, N)$ and we analyse transitions from $(\Gamma, E[M, \Gamma])$ where M is in evaluation position.

1. If M is not a value and $M \longmapsto_\mathsf{n} M'$, then the only transition is $(\Gamma, E[M, \Gamma]) \xrightarrow{\tau} (\Gamma, E[M', \Gamma])$ which is trivially answered to, using $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$, yielding processes in $\mathcal{C}_\mathsf{e}(\mathcal{S})$.

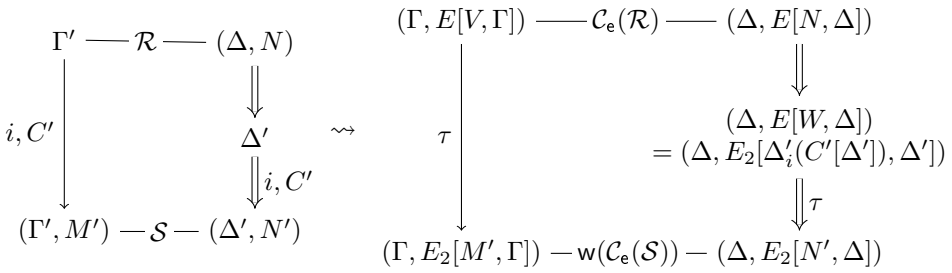
2. If M is a value V, we write $\Gamma' = (\Gamma, V)$ and $E[V, \Gamma] = C[\Gamma']$ for some C. We follow an analysis similar to the one for $\mathcal{C}$ in Lemma 5.3.10, using weakening to remove the extra value when needed. Using Lemma 5.3.5 we get $(\Delta, N) \Longrightarrow (\Delta, W) \triangleq \Delta'$ such that $\Gamma'\ \mathcal{S}_1\ \Delta'$ with $\mathcal{S}_1 \triangleq \mathbf{wb}'(f_{\gtrsim}(\mathcal{S}))$.

   (a) If $C[\Gamma']$ is not a value, then:

      i. if $C[\Gamma'] \xrightarrow{\tau} C'[\Gamma']$ with $C[\Delta'] \xrightarrow{\tau} C'[\Delta']$, then we need to relate $(\Gamma, C'[\Gamma'])$ to $(\Delta, C'[\Delta'])$:

$$\dfrac{\dfrac{\Gamma'\ \ \mathcal{S}_1\ \ \Delta'}{(\Gamma', C'[\Gamma'])\ \ \mathcal{C}(\mathcal{S}_1)\ \ (\Delta', C'[\Delta'])}}{(\Gamma, C'[\Gamma'])\ \ \mathsf{w}(\mathcal{C}(\mathcal{S}_1))\ \ (\Delta, C'[\Delta'])}$$

      ii. otherwise $C[\Gamma'] = E_2[\Gamma_i'(C'[\Gamma']), \Gamma']$ for some E_2 and C' and the transition is of the form $(\Gamma, E_2[\Gamma_i'(C'[\Gamma']), \Gamma']) \xrightarrow{\tau} (\Gamma, E_2[M', \Gamma'])$ which corresponds to the transition $\Gamma' \xrightarrow{i,C'} (\Gamma', M')$, to which RHS responds with $(\Delta, N) \Longrightarrow \Delta' \xRightarrow{i,C'} (\Delta', N')$ which allows us to close the diagram with $\mathsf{w}(\mathcal{C}_\mathsf{e}(\mathcal{S}))$ (using w to get Γ and Δ from Γ' and Δ').



   (b) or $E[V, \Gamma] = C[\Gamma']$ is a value $C_v[\Gamma']$ (we ignore the case where $C_v = [\cdot]_i$ for which we progress to $\mathsf{str}(\mathcal{S})$) and the transition is $(\Gamma, C_v[\Gamma']) \triangleq \Gamma'' \xrightarrow{i,C} (\Gamma'', M')$ with $\Gamma_i''(C[\Gamma'']) \longmapsto_\mathsf{n} M'$. Again there are two cases depending on i:

      i. if $i \leq |\Gamma|$ then $\Gamma_i''(C[\Gamma'']) = \Gamma_i'(C'[\Gamma'])$ for some C' and we use transition i, C' with $\mathcal{R} \overset{w}{\rightarrowtail} \mathcal{S}$ to get $(\Delta, N) \xRightarrow{i,C'} (\Delta', N')$ such that $\Delta_i''(C[\Delta'']) \Longmapsto_\mathsf{n} N'$ (noting $\Delta'' = (\Delta, C_v[\Delta'])$) so we get the transition $(\Delta, E[N, \Delta]) \Longrightarrow (\Delta, E[W, \Delta]) = \Delta'' \xRightarrow{i,C} (\Delta'', N')$. Now from $(\Gamma', M')\ \mathcal{S}\ (\Delta', N')$ we relate $(\Gamma, C_v[\Gamma'], M')$ to $(\Delta, C_v[\Delta'], N')$:

$$\dfrac{\dfrac{(\Gamma', M')\ \ \mathcal{S}\ \ (\Delta', N')}{(\Gamma', C_v[\Gamma'], M')\ \ \mathsf{str}(\mathcal{S})\ \ (\Delta', C_v[\Delta'], N')}}{(\Gamma, C_v[\Gamma'], M')\ \ \mathsf{w}(\mathsf{str}(\mathcal{S}))\ \ (\Delta, C_v[\Delta'], N')}$$

      ii. if $i = |\Gamma| + 1$ then $M' = C_2[\Gamma']$ for some C_2 and we can get the same transition from the RHS: $(\Delta, E[N, \Delta]) \Longrightarrow (\Delta, C_v[\Delta']) \xrightarrow{i,C} (\Delta, C_v[\Delta'], C_2[\Delta'])$ to which we can relate (Γ'', M') in three steps:

$$\dfrac{\dfrac{\dfrac{\Gamma'\ \ \mathcal{S}_1\ \ \Delta'}{(\Gamma', C_2[\Gamma'])\ \ \mathcal{C}(\mathcal{S}_1)\ \ (\Delta', C_2[\Delta'])}}{(\Gamma', C_v[\Gamma'], C_2[\Gamma'])\ \ \mathsf{str}(\mathcal{C}(\mathcal{S}_1))\ \ (\Delta', C_v[\Delta'], C_2[\Delta'])}}{(\Gamma, C_v[\Gamma'], C_2[\Gamma'])\ \ \mathsf{w}(\mathsf{str}(\mathcal{C}(\mathcal{S}_1)))\ \ (\Delta, C_v[\Delta'], C_2[\Delta'])}$$

□

Once more, the compatibility up to of the up to context functions entails the substitutivity properties of environmental bisimilarity. In [SKS11] the two aspects (substitutivity and up to context) had to be proved

separately, with similar proofs. Moreover the two cases of contexts (arbitrary contexts and evaluation contexts) had to be considered at the same time, within the same proof. Here, in contrast, the machinery of compatible functions allows us to split the proof into simpler proofs.

Remark 5.3.12. A transition system ensuring full abstraction as in Theorem 5.3.2 does not guarantee the compatibility of the up-to techniques specific to the language being considered. For instance, a simpler and maybe more natural alternative to the second transition in (5.5) is the following one:

$$\frac{}{\Gamma \xrightarrow{i,C} (\Gamma, \Gamma_i(C[\Gamma]))} \ . \tag{5.6}$$

With this rule, full abstraction holds, but up to context is unsound: for any Γ and Δ, the singleton relation $\{(\Gamma, \Delta)\}$ is a bisimulation up to $\mathcal{C}$: indeed, using rule (5.6), the derivatives of the pair Γ, Δ are of the shape $\Gamma_i(C[\Gamma])$, $\Delta_i(C[\Delta])$, and they can be discarded immediately, up to context $[\cdot]_i C$. If up to context were sound then we would deduce that any two terms are bisimilar. (The rule in (5.5) prevents such a behaviour since it ensures that the tested values are 'consumed' immediately.)

Beta-reduction is included in expansion

Lemma 5.3.15 shows that the full β reduction, $\to_\beta$ (whereby an arbitrary redex in a term is consumed) is an expansion. This is a useful property in applications of up-to techniques. The proof of the lemma requires results of confluence between $\to_\beta$ and $\longmapsto_{\mathsf{n}}$, and the standardisation theorem [Bar84].

Lemma 5.3.13. $(\leftarrow_{\mathsf{n}} \circ \to_\beta^*) \subseteq (\to_\beta^* \circ \leftarrow_{\mathsf{n}}^=)$.

Proof. Suppose $M \longmapsto_{\mathsf{n}} M'$ and $M \to_\beta M''$, then M, M', and M'' are of the following forms, where $u_i \to_\beta^= u'_i$ for all i (we write for later usage the unifying term N):

$$\begin{array}{ll} M & = (\lambda x.u_1) u_2 u_3 \dots u_n \\ M'' & = (\lambda x.u'_1) u'_2 u'_3 \dots u'_n \text{ or } M'' = M' \\ M' & = u_1[u_2/x]\ u_3 \dots u_n \\ N & = u'_1[u'_2/x]\ u'_3 \dots u'_n \end{array}$$

depending on the position of the β-redex of $M \to_\beta M''$. In the case $M'' = M'$ the confluence is trivial. Otherwise, we see as well $M'' \longmapsto_{\mathsf{n}} N$ and $M' \to_\beta^* N$. We supposed $M \to_\beta M''$ instead of $M \to_\beta^* M''$: we conclude by induction. □

Lemma 5.3.14. *If $M \to_\beta^* V$ and V is a value, then for some value W, $M \longmapsto_{\mathsf{n}}^* W$.*

Proof. By the standardisation theorem (Theorem 11.4.7 from [Bar84]), there is a standard reduction $M \to_s^* V$ from M to V. Since "being a value" is preserved by β-reduction, there is a first W such that $M \to_s^* W \to_s^* V$. We prove $M \longmapsto_{\mathsf{n}}^* W$. Inducting on the first $\to_s^*$ we need only to prove that in a standard reduction from a non-value to a value, the first step is a call-by-name step.

From a non-value term $N = (\lambda x.u_1) u_2 u_3 \dots u_n$ (with $n \geq 2$), consider the standard reduction $N \to_s N' \to_s^* W$ and let us prove $N \longmapsto_{\mathsf{n}} N'$. Let δ be the cbn redex[2] $(\lambda x.u_1) u_2$. In the standard reduction there must be a redex δ_j residual of δ, otherwise W would be of the form $(\lambda x.u'_1) u'_2 u'_3 \dots u'_n$ with $u_i \to_\beta^* u_i$ and W would not be a value. Let δ_1 be the first redex. Suppose by contradiction that $\delta_1 \neq \delta$. Then:

1. either δ_1 is in u_1 or in u_2: then δ contains δ_1,
2. or δ_1 is in u_i for some $i \geq 3$: then δ is to the left of δ_1.

In both cases, δ_1 and δ are in forbidden configurations for a standard reduction as δ_j (residual of δ) is triggered after δ_1. We conclude $\delta_1 = \Delta$, i.e. that the first redex of a standard reduction to a value is a cbn redex. □

[2] In [Bar84] the notation for redexes is Δ instead of δ.

Lemma 5.3.15. *If $M \to_\beta N$ then $(\Gamma, M) \gtrsim (\Gamma, N)$.*

Proof. We prove $\mathcal{R} = \{(\Gamma, M), (\Delta, N) \mid M \to_\beta^* N \ \wedge\ \forall i\ \Gamma_i \to_\beta^* \Delta_i\}$ is an expansion relation. Let us play the transitions for either sides of $M \to_\beta^* N$. In the following, we write $V\{N\}$ for M' when $VN \longmapsto_{\mathsf{n}} M'$ (i.e. $(\lambda x.P)\{N\} \triangleq P\{N/x\}$). We distinguish cases for silent and visible transitions, and transitions from the LHS and from the RHS:

1. $(\Gamma, M) \xrightarrow{\tau} (\Gamma, M')$, i.e. $M \longmapsto_{\mathsf{n}} M'$. Using Lemma 5.3.13 we find N' such that $M' \to_\beta^* N'$ and $N \longmapsto_{\mathsf{n}}^= N'$, i.e. $(\Delta, N) \xrightarrow{\hat{\tau}} (\Delta, N')$.

2. $(\Delta, N) \xrightarrow{\tau} (\Delta, N')$, i.e. $N \longmapsto_{\mathsf{n}} N'$. Obviously N is not a value, hence neither is M so for some M', $M \longmapsto_{\mathsf{n}} M'$. By Lemma 5.3.13 for some N'' we have $M' \to_\beta^* N''$ and $N \longmapsto_{\mathsf{n}}^= N''$, so:

   (a) either $N = N''$ in which case $M' \to_\beta^* N \to_\beta N'$,

   (b) or $N \longmapsto_{\mathsf{n}} N'$, hence $N' = N''$ (by determinism of $\longmapsto_{\mathsf{n}}$) and $M' \to_\beta^* N'$.

   In both cases, $(\Gamma, M) \overset{\tau}{\Longrightarrow} (\Gamma, M')$ and $(\Gamma, M')\ \mathcal{R}\ (\Delta, N')$.

3. M is a value V and $(\Gamma, V) \triangleq \Gamma' \xrightarrow{i,C} (\Gamma', \Gamma_i'\{C\Gamma'\})$ for some context C. Then N, as a β-derivative of a value, is a value W and $\Delta' \triangleq (\Delta, W) \xrightarrow{i,C} (\Delta', \Delta_i'\{C\Delta'\})$. In passing note that $\Gamma'\ \mathcal{R}\ \Delta'$. Remarking that $\Gamma_i'(C\Gamma') \longmapsto_{\mathsf{n}} \Gamma_i'\{C\Gamma'\}$ and $\Delta_i'(C\Delta') \longmapsto_{\mathsf{n}} \Delta_i'\{C\Delta'\}$ and $\Gamma_i'(C\Gamma') \to_\beta^* \Delta_i'(C\Delta')$ by congruence of $\to_\beta^*$, we conclude by Lemma 5.3.13.

4. N is a value W and $(\Delta, W) \triangleq \Delta' \xrightarrow{i,C} (\Delta', \Delta_i'\{C\Delta'\})$. Then by Lemma 5.3.14 $M \longmapsto_{\mathsf{n}}^* V$ for some value V (plus $V \to_\beta^* W$ by Lemma 5.3.13), which means $(\Gamma, M) \Longrightarrow (\Gamma, V) \triangleq \Gamma'$ and incidentally $\Gamma'\ \mathcal{R}\ \Delta'$.

   Now from Γ' we do the transition $\Gamma' \xrightarrow{i,C} (\Gamma', \Gamma_i'\{C\Gamma'\})$ and we relate the two resulting configurations as in the item before.

From left to right, we answered to $\xrightarrow{\tau}$ with $\xrightarrow{\hat{\tau}}$ and $\xrightarrow{i,C}$ with $\xrightarrow{i,C}$, and from right to left, $\xrightarrow{\tau}$ with $\xrightarrow{\tau}$ and $\xrightarrow{i,C}$ with $\Longrightarrow\xrightarrow{i,C}$ so $\mathcal{R}$ is indeed an expansion relation. □

Lemma 5.3.15 is a useful term rewriting tool for bisimulation proofs, to reduce redexes in depth. Note that rewriting at toplevel corresponds to call-by-name reductions (proving $\longmapsto_{\mathsf{n}} \subseteq \gtrsim$ is much easier as $\longmapsto_{\mathsf{n}}$ is deterministic).

5.4 Imperative call-by-value λ-calculus

In this section we study the addition of imperative features (higher-order references, that we call locations), to a call-by-value λ-calculus. It is known that finding powerful reasoning techniques for imperative higher-order languages is a hard problem. The language, ΛR, is a simplified variant of that in [KW06, SKS11]. The syntax of terms, values, and evaluation contexts, as well as the reduction semantics are given in Figure 5.6. A λ-term M is run in a *store*: a partial function from locations to closed values, whose domain includes all free locations of both M and its own co-domain. We use letters s, t to range over stores. New store locations may be created using the operator $\nu\ell\, M$; the content of a store location ℓ may be rewritten using $\mathsf{set}_\ell V$, or read using $\mathsf{get}_\ell V$ (the former instruction returns a value, namely the identity $I \triangleq \lambda x.x$, and the argument of the latter one is ignored). We denote the reflexive and transitive closure of $\longmapsto_{\mathsf{R}}$ by $\Longmapsto_{\mathsf{R}}$.

Note that in contrast with the languages in [KW06, SKS11], locations are not directly first-class values; the expressive power is however the same: a first-class location ℓ can always be encoded as the pair $(\mathsf{get}_\ell, \mathsf{set}_\ell)$.

We present the first-order LTS for ΛR, and then we relate the resulting strong and weak bisimilarities directly with contextual equivalence (the reference equivalence in λ-calculi). Alternatively, we could have

$$M ::= x \mid MM \mid \nu\ell\, M \mid V \qquad V ::= \lambda x.M \mid \mathsf{set}_\ell \mid \mathsf{get}_\ell \qquad E ::= [\cdot] \mid EV \mid ME$$

$$\frac{}{(s;(\lambda x.M)V) \longmapsto_{\mathsf{R}} (s; M\{V/x\})} \qquad \frac{\ell \notin \mathsf{dom}(s)}{(s;\nu\ell\, M) \longmapsto_{\mathsf{R}} (s[\ell \mapsto I]; M)} \qquad \frac{\ell \in \mathsf{dom}(s)}{(s;\mathsf{get}_\ell V) \longmapsto_{\mathsf{R}} (s; s[\ell])}$$

$$\frac{\ell \in \mathsf{dom}(s)}{(s;\mathsf{set}_\ell V) \longmapsto_{\mathsf{R}} (s[\ell \mapsto V]; I)} \qquad \frac{(s;M) \longmapsto_{\mathsf{R}} (s';M')}{(s;E[M]) \longmapsto_{\mathsf{R}} (s';E[M'])}$$

Figure 5.6: The imperative λ-calculus

related the first-order bisimilarities to the environmental bisimilarities of ΛR, and then inferred the correspondence with contextual equivalence from known results about environmental bisimilarity, as we did for ΛN.

We write $(s;M)\downarrow$ when M is a value; and $(s;M)\Downarrow$ if $(s;M) \Longmapsto_{\mathsf{R}}\downarrow$. For the definition of contextual equivalence, we distinguish the cases of values and of arbitrary terms, because they have different substitutivity properties: values can be tested in arbitrary contexts, while arbitrary terms must be tested only in evaluation contexts—the two notions coincide on values. As in [SKS11], we consider contexts that do not contain free locations (they can contain bound locations). We refer to [SKS11] for more details on these aspects.

Definition 5.4.1.
- For values V, W, we write $(s;V) \equiv (t;W)$ when $(s;C[V])\Downarrow$ iff $(t;C[W])\Downarrow$, for all location-free context C.
- For terms M and N, we write $(s;M) \equiv (t;N)$ when $(s;E[M])\Downarrow$ iff $(t;E[N])\Downarrow$, for all location-free evaluation context E.

We now define ΛR^1, the first-order LTS for ΛR. The states and the transitions for ΛR^1 are similar to those for the pure λ-calculus of Section 5.3, with the addition of a component for the store:

$$\mathit{Pr} = \{(s;\Gamma,M) \mid \Gamma \text{ sequence of closed values, } M \text{ a closed term, } s \text{ a store}\}\ .$$

The transitions are similar too, but the visible label is adapted in two ways. First, the argument context now is a value context, since we are in a call-by-value setting. Second, we also need to add a list of value contexts to the transition, in order to account for additional store information:

$$\mathit{Act} : \mu \ ::= \ \tau \ \mid \ i, C_v, (C_v^1, \ldots, C_v^n)\ .$$

The two transitions (5.5) of call-by-name λ-calculus become:

$$\frac{(s;M) \longmapsto_{\mathsf{R}} (s';M')}{(s;\Gamma,M) \xrightarrow{\tau} (s';\Gamma,M')} \qquad \frac{\Gamma' = \Gamma, \mathsf{getset}(r) \qquad (s \uplus r[\Gamma'];\Gamma_i(C_v[\Gamma'])) \longmapsto_{\mathsf{R}} (s';M')}{(s;\Gamma) \xrightarrow{i,C_v,\mathsf{cod}(r)} (s';\Gamma',M')}$$

The first rule is the analogous of the first rule in (5.5). The important differences are in the second rule. First, since we are *call-by-value*, C_v now ranges over $\mathbb{C}_v$, the set of *value contexts* (i.e., contexts of the form $\lambda x.C'$) without free locations. Moreover, since we are now *imperative*, in a transition we must permit the creation of new locations, and a term supplied by the environment should be allowed to use them. In the rule, the new store is represented by r (whose domain has to be disjoint from that of s). Correspondingly, to allow manipulation of these locations by the observer, for each new location ℓ we make set_ℓ and get_ℓ available, as an extension of the environment; in the rule, these are collectively written $\mathsf{getset}(r)$, and Γ' is the extended environment. Finally, we must initialise the new store, using terms that are constructed using the extended environment Γ'; that is, each new location ℓ is initialised with a term $D_\ell[\Gamma']$ (for $D_\ell \in \mathbb{C}_v$). Moreover, the

contexts D_ℓ chosen must be made visible in the label of the transition. To take care of these aspects, we view r as a *store context*, i.e. a tuple of assignments $\ell \mapsto D_\ell$. Thus the initialisation of the new locations is written $r[\Gamma']$; and, denoting by $\mathtt{cod}(r)$ the tuple of the contexts D_ℓ in r, we add $\mathtt{cod}(r)$ to the label of the transition. Note also that, although C_v and D_ℓ are location-free, their holes may be instantiated with terms involving the set_ℓ and get_ℓ operators, and these allow manipulation of the store.

Once more, on the (strong and weak) bisimilarities that are derived from this first-order LTS we can import the theory of compatible functions and bisimulation enhancements. Concerning additional up-to functions, specific to ΛR, the functions w, str, $\mathcal{C}$ and $\mathcal{C}_\mathsf{e}$ are adapted from Section 5.3 in the expected manner—contexts C_v, C and E must be location-free. A further function for ΛR is store, which manipulates the store by removing locations that do not appear elsewhere (akin to garbage collection); thus, $\mathsf{store}(\mathcal{R})$ is the set of all pairs

$$((s \uplus r[\Gamma'];\Gamma', M),\ (t \uplus r[\Delta'];\Delta', N))$$

such that $(s;\Gamma,M)\ \mathcal{R}\ (t;\Delta,N)$, and with $\Gamma' = \Gamma, \mathsf{getset}(r)$ and $\Delta' = \Delta, \mathsf{getset}(r)$.

In practice s and t might have different domains, so we may want to have two separate r and r' for the LHS and RHS, such that $\mathtt{cod}(r) = \mathtt{cod}(r')$. Instead, we chose to reason up to renaming of locations (and impose $r = r'$) which corresponds indeed to a compatible function. Alternatively, we can work up to $\sim$, since renaming locations preserves strong bisimilarity. (This case is simpler than the "up to injective substitution" technique in the π-calculus, since names appear in transitions but locations do not.)

Lemma 5.4.2. *The set* $\{\mathsf{w}, \mathsf{str}, \mathcal{C}_\mathsf{e}, \mathsf{store}, \mathcal{C}\}$ *is*

- $\mathbf{b}$*-compatible up to the identity function,*
- $\mathbf{e}$*-compatible up to* $\mathbf{e}' \cup (\mathcal{R} \mapsto \gtrsim\mathcal{R}\sim)$.
- $\mathbf{wb}$*-compatible up to* $\mathbf{wb}' \cup f_{\gtrsim}$.

Proof. In each case, we apply the same methodology as in Theorem 5.3.8, with more technical details to be handled, as the store is to be accounted for. However, we provide details mainly for the evolution starting from store itself, which is the most interesting and the only one fundamentally new. We explain how the others are modified, compared to the proof of Theorem 5.3.8.

In fact, we handle store first, because of the naming convention we will use: we write Γ^V for Γ, V and Γ^r for $\Gamma, \mathsf{getset}(r)$. This way, when $(s,\Gamma) \xrightarrow{i,C,\mathtt{cod}(r)} (s' \uplus r[\Gamma^r], \Gamma^r, M')$ when $(s;\Gamma_i(C[\Gamma^r])) \longmapsto_\mathsf{R} (s';M')$.

Now store can be defined as $(s \uplus r[\Gamma^r];\Gamma^r,M)\ \mathsf{store}(\mathcal{R})\ (t \uplus r[\Delta^r];\Delta^r,N)$ when $(s;\Gamma,M)\ \mathcal{R}\ (t;\Delta,N)$. We analyse transitions of the former. Silent transitions are one again easy to handle, as the locations of M are contained in the domain of s, the other part $r[\Gamma^r]$ left unchanged (progressing to $\mathsf{store}(\mathcal{S})$). We handle now visible transitions from the LHS (then M is a value V), labelled by μ such that $\mu = i, C, \mathtt{cod}(u)$ for some u (we suppose for simplicity that the locations used by u are fresh). The transition satisfies:

$$\frac{(s \uplus r[\Gamma^r] \uplus u[\Gamma^{rVu}];\Gamma_i^{rV}(C[\Gamma^{rVu}])) \longmapsto_\mathsf{R} (s';M')}{(s \uplus r[\Gamma^r];\Gamma^{rV}) \xrightarrow{i,C,\mathtt{cod}(u)} (s';\Gamma^{rVu},M')} \tag{5.7}$$

There are two cases, depending if Γ_i^{rV} is in Γ^V or $\mathsf{getset}(r)$.

1. Suppose $i \leq |\Gamma|$ or $i = |\Gamma^{rV}| + 1 = |\Gamma| + 2|r| + 1$. Then $\Gamma_i^{rV} = \Gamma_{i'}^V$ for some i', and we can in fact run the same $\longmapsto_\mathsf{R}$ transition from (s,Γ^V), using label $\mu' = i', C', \mathtt{cod}(v)$ for some v and C' such that:
   (a) $r[\Gamma^r] \uplus u[\Gamma^{rVu}] = v[\Gamma^{Vv}]$
   (b) $C[\Gamma^{rVu}] = C'[\Gamma^{Vv}]$

   The premise of (5.7) can be rewritten to the following, with a different conclusion:

$$\frac{(s \uplus v[\Gamma^{Vv}];\Gamma_{i'}^V(C'[\Gamma^{Vv}])) \longmapsto_\mathsf{R} (s';M')}{(s;\Gamma^V) \xrightarrow{i',C',\mathtt{cod}(v)} (s';\Gamma^{Vv},M')}\ .$$

We can derive the corresponding (weak or not) transition labelled with $i', C', \mathtt{cod}(v)$ from the RHS $(t; \Delta, N)$ which will reduce to $(t'; \Delta^W)$ and then $(t''; \Delta^{Vv}, N')$ knowing that $(s'; \Gamma^{Vv}, M')\ \mathcal{S}\ (t''; \Delta^{Vv}, N')$. We can then replace Γ^{Vv} and Δ^{Wv} with Γ^{rVu} and Δ^{rWu} up to permutation.

2. Suppose $i \in \{|\Gamma|+1, \dots, |\Gamma|+2|r|\}$. Then Γ_i^{rV} is either get_ℓ or set_ℓ with $\ell \in \mathsf{dom}(r)$.

   (a) if $\Gamma_i^{rV} = \mathsf{get}_\ell$ then s' is not modified and $M' = r[\Gamma^r]_\ell$ is a context of Γ^r and hence of Γ^{Vv} using the same v as above. Hence the transition goes to $(s \uplus v[\Gamma^{Vv}]; \Gamma^{Vv}, C_1[\Gamma^{Vv}]) \triangleq x'$. Using Lemma 5.3.5 we get $(t; \Delta, N) \Longrightarrow (t'; \Delta^W)$ related to (s, Γ^V) through $\mathcal{S}_1 = \mathbf{wb}'(\gtrsim \mathcal{S} \lesssim)$.
   Note that when running $\mathbf{e}$ progressions instead of $\mathbf{wb}$, we use Lemma 5.3.7 when N is a value W to get $(s; M)$ to reach $(s'; V)$ with $((s'; \Gamma, V), (t; \Delta, W)) \in \mathbf{e}'(\gtrsim \mathcal{S})$.
   Finally we can relate x' to $(t' \uplus v[\Delta^{Wv}]; \Delta^{Wv}, C_1[\Delta^{Wv}])$ through $\mathcal{C}(\mathsf{store}(\mathcal{S}_1))$.

   (b) if $\Gamma_i^{rV} = \mathsf{set}_\ell$ then s' is modified at $\ell \in \mathsf{dom}(r)$ (so we just have to change of r) and $M' = I = C_1[\Gamma^{Vv}]$ for a trivial C_1, so we progress again, using the same notations as before, to $\mathcal{C}(\mathsf{store}(\mathcal{S}_1))$.

Finally we get that $\mathsf{store} \overset{b}{\rightsquigarrow} \mathsf{store} \cup \mathcal{C} \circ \mathsf{store} \circ g_b$ with:

$$g_{\mathbf{b}} \triangleq \mathcal{S} \mapsto \mathcal{S} \qquad g_{\mathbf{wb}} \triangleq \mathcal{S} \mapsto \mathbf{wb}'(\gtrsim \mathcal{S} \lesssim) \qquad g_{\mathbf{e}} \triangleq \mathcal{S} \mapsto \mathbf{e}'(\gtrsim \mathcal{S} \sim)\ .$$

We will make reference to g_b in the evolutions for the other functions:

1. We prove $\mathsf{w} \overset{b}{\rightsquigarrow} \mathsf{w}$ along the same lines as for Lemma 5.3.3.

2. $\mathsf{str} \overset{b}{\rightsquigarrow} \mathsf{str} \cup \mathcal{C} \circ \mathsf{store} \circ \mathsf{str} \circ g_b$: we analyse the transitions of $(s; \Gamma, C_v[\Gamma], M)$.

   (a) As long as M is not a value we stay in $\mathsf{str}(\mathcal{S})$.

   (b) If M is a value V, and we run V itself or any of the Γ_i, then we run the same transition $i, C, \mathtt{cod}(r)$ on the original configuration $(s; \Gamma, V)$ and we still stay in $\mathsf{str}(\mathcal{S})$.

   (c) If however we run $C_v[\Gamma]$ we have to use relation $\mathcal{S}_1 = g_b(\mathcal{S})$ to get $(t; N)$ to reach a value $(t'; W)$ with $((s; \Gamma, V), (t'; \Delta, W)) \in \mathcal{S}_1$ and then we find ourselves with a completely arbitrary context $C_2[\Gamma]$ (we use $\mathcal{C}$) and with a augmented store (we use store) to end up in a relation built over $\mathcal{S}_1$ (writing $\Gamma' = \Gamma, C_v[\Gamma], V$ and $\Delta' = \Delta, C_v[\Delta], W$):

$$\frac{\dfrac{\dfrac{(s; \Gamma^V)\ \mathcal{S}_1\ (t'; \Delta^W)}{(s; \Gamma')\ \mathsf{str}(\mathcal{S}_1)\ (t'; \Delta')}}{(s \uplus r[\Gamma'^r]; \Gamma'^r)\ \mathsf{store}(\mathsf{str}(\mathcal{S}_1))\ (t' \uplus r[\Delta'^r]; \Delta'^r)}}{(s \uplus r[\Gamma'^r]; \Gamma'^r, C_2[\Gamma])\ \mathcal{C}(\mathsf{store}(\mathsf{str}(\mathcal{S}_1)))\ (t' \uplus r[\Delta'^r]; \Delta'^r, C_2[\Gamma])}\ .$$

3. $\mathcal{C} \overset{b}{\rightsquigarrow} f_1 \triangleq \mathsf{str} \cup \mathcal{C} \circ \mathsf{store} \circ \mathsf{str} \cup \mathcal{C} \cup \mathcal{C}_{\mathsf{e}} \cup \mathsf{w} \circ \mathcal{C} \circ \mathsf{store}$: we analyse the transitions of $(s; \Gamma, C[\Gamma])$.

   (a) If $C[\Gamma]$ is a value, then it is a particular case of str, progressing to $\mathsf{str}(\mathcal{S}) \cup \mathcal{C}(\mathsf{store}(\mathsf{str}(\mathcal{S})))$ (this time, $\mathcal{S}$ instead of $g_b(\mathcal{S})$ since we already have values).

   (b) Otherwise, we analyse the τ transitions of $C[\Gamma]$: if it is a reduction between two proper value contexts of Γ we just stay in $\mathcal{C}(\mathcal{R})$.

   (c) If some Γ_i is run (i.e. $C[\Gamma] = E[\Gamma_i(C'_v[\Gamma]), \Gamma]$), we can run it starting from the original configuration using the label $i, C'_v, \emptyset$ using the evaluation context function, progressing to $\mathcal{C}_{\mathsf{e}}(\mathcal{S})$.

   (d) The most interesting part is when $C[\Gamma]$ creates a private location: $C[\Gamma] = E[\nu\ell\, C_1[\Gamma], \Gamma]$ and $(s; \Gamma, C[\Gamma]) \xrightarrow{\tau} (s \uplus [\ell \mapsto I]; \Gamma, D[\Gamma, \mathsf{getset}(\ell)])$ for some context D (indeed, $C_1\Gamma$ can contain some get_ℓ and set_ℓ). In fact we prove the stronger result (using weakening w) that the resulting

configurations with the context D (using $\mathcal{C}$) are still related if get_ℓ's and set_ℓ's are available (using store). Below, we write Λ for $\mathsf{getset}(\ell)$.

$$\dfrac{\dfrac{\dfrac{(s;\Gamma)\ \ \mathcal{R}\ \ (t;\Delta)}{(s\uplus[\ell\mapsto I];\Lambda,\Gamma)\ \ \mathsf{store}(\mathcal{R})\ \ (t\uplus[\ell\mapsto I];\Lambda,\Delta)}}{(s\uplus[\ell\mapsto I];\Gamma,\Lambda,D[\Gamma,\Lambda])\ \ \mathcal{C}(\mathsf{store}(\mathcal{R}))\ \ (t\uplus[\ell\mapsto I];\Delta,\Lambda,D[\Delta,\Lambda])}}{(s\uplus[\ell\mapsto I];\Gamma,D[\Gamma,\Lambda])\ \ \mathsf{w}(\mathcal{C}(\mathsf{store}(\mathcal{R})))\ \ (t\uplus[\ell\mapsto I];\Delta,D[\Delta,\Lambda])}\ .$$

4. $\mathcal{C}_\mathsf{e} \overset{b}{\rightsquigarrow} \mathcal{C}_\mathsf{e} \cup f_1 \circ g_b$: we look into the transitions of $(s;\Gamma,E[M,\Gamma])$.
   (a) If M is not a value, we trivially progress to $\mathcal{C}_\mathsf{e}(\mathcal{S})$.
   (b) If M is a value, we are in a special case of the analysis we did for $\mathcal{C}$, except that we use $g_b(\mathcal{S})$ instead of $\mathcal{R}$, so we progress to $f_1(g_b(\mathcal{S}))$ where f_1 is defined in the proof for $\mathcal{C}$.

This evolutions hold for $\mathbf{wb}$ (up to $\mathbf{wb}' \cup g_{\mathbf{wb}}$), $\mathbf{e}$ (up to $\mathbf{e}' \cup g_\mathbf{e}$), and $\mathbf{b}$. Note that the shape of the weak transitions (let it be $\overset{\mu}{\Longrightarrow}$, $\overset{\hat{\mu}}{\Longrightarrow}$ or $\overset{\hat{\mu}}{\longrightarrow}$) is not central in the proof, apart from the usage of the function g_b using Lemmas 5.3.5 and 5.3.7. □

The techniques $\mathcal{C}$ and $\mathcal{C}_\mathsf{e}$ allow substitutivity using location-free contexts, from which we can derive the soundness part of Theorem 5.4.3.

Theorem 5.4.3. $(s;M) \equiv (t;N)$ *iff* $(s;M) \approx (t;N)$.

Proof. ($\Leftarrow$) **Soundness** follows from congruence: suppose $(s;M) \approx (t;N)$. Let E be an evaluation context. Since $\mathcal{C}_\mathsf{e}$ is compatible up to (Lemma 5.4.2) by Lemma 5.1.8 we know $\approx$ is a $\mathcal{C}_\mathsf{e}$-congruence, hence $(s;E[M]) \approx (t;E[N])$.

Suppose now $(s;E[M])$ reaches a value. Then we can derive a visible transition from it, and by weak bisimulation we can derive the same transition from $(t;E[N])$, meaning $E[N]$ also reaches a value. This means $(s;E[M])\Downarrow$ implies $(t;E[N])\Downarrow$. The same way, $(t;E[N])\Downarrow$ implies $(s;E[M])\Downarrow$.

($\Rightarrow$) **Completeness** is obtained by standard means. We prove that the relation $\mathcal{R}$ below is a weak bisimulation (E ranges over location-free evaluation contexts):

$$\mathcal{R} \triangleq \left\{((s;\Gamma,M),(t;\Delta,N))\ \text{s.t.}\ \forall E\ (s;E[M,\Gamma])\Downarrow \text{iff}\ (t;E[N,\Delta])\Downarrow\right\}\ . \tag{5.8}$$

Since $\mathcal{R}$ is symmetric, we only look at the transitions $(s;\Gamma,M) \xrightarrow{\mu}$ which we catch up with transition $(t;\Delta,N) \overset{\hat{\mu}}{\Longrightarrow}$ leading to $\mathcal{R}$-related processes.

When $\mu = \tau$ we have $(s;M) \longmapsto_\mathsf{R} (s';M)$ we can easily see that $(s;E[M,\Gamma])\Downarrow$ iff $(s';E[M',\Gamma])\Downarrow$ by determinism of $\longmapsto_\mathsf{R}$, so we have in fact $(s';\Gamma,M')\ \mathcal{R}\ (t;\Delta,N)$ and the transition $\xrightarrow{\tau}$ is answered with no transition at all.

We now suppose that μ is visible, i.e. $\mu = i,C,\mathsf{cod}(r)$ and $(s;\Gamma,M) \xrightarrow{\mu} (s';\Gamma'',M')$ for some s', Γ'', M' satisfying $(*)$ below. It also means that M is a value V and thus $(s;M)\Downarrow$. By choosing $E = [\cdot]_1$ and $u = \emptyset$ in (5.8) we know that $(t;N)\Downarrow$ and thus $(t;N) \Longmapsto_\mathsf{R} (t';W)$ for some value W. Then, we get the weak transition $(t;\Delta,N) \overset{\mu}{\Longrightarrow} (t'';\Delta'',N')$ through $(t';\Delta,W)$, satisfying $(**)$ below.

$$\begin{array}{llll} \Gamma' = \Gamma,V & \Gamma'' = \Gamma',\mathsf{getset}(r) & (r[\Gamma''] \uplus s;\Gamma'_i(C[\Gamma''])) \longmapsto_\mathsf{R} (s';M') & (*) \\ \Delta' = \Delta,W & \Delta'' = \Delta',\mathsf{getset}(r) & (r[\Delta''] \uplus t';\Delta'_i(C[\Delta''])) \longmapsto_\mathsf{R} (t'';N') & (**) \end{array}$$

We prove now $(s';\Gamma'',M')\ \mathcal{R}\ (t'';\Delta'',N')$. Let E be any location-free evaluation context, we prove:

$$(s';E[M',\Gamma'']) \Downarrow \quad \text{iff} \quad (t'';E[N',\Delta'']) \Downarrow\ . \tag{5.9}$$

Backtracking one step using $(*)$ and $(**)$ it is enough to prove

$$(r[\Gamma''] \uplus s; E[\Gamma'_i(C[\Gamma'']), \Gamma'']) \Downarrow \quad \text{iff} \quad (r[\Gamma''] \uplus t'; E[\Delta'_i(C[\Delta'']), \Delta'']) \Downarrow \tag{5.10}$$

which we do by exhibiting an evaluation context F for which we already have (by instantiating the definition of $\mathcal{R}$ with F) the equation (5.11) below, and we also know that each member of (5.10) is a derivative of the corresponding member of (5.11).

$$(s; F[M, \Gamma]) \Downarrow \quad \text{iff} \quad (t; F[N, \Delta]) \Downarrow \tag{5.11}$$

For (5.11) to imply (5.10), we instantiate F as follows:

$$F \triangleq \texttt{let}\ x = [\cdot]_1\ \texttt{in}\ \nu\ell_1 \ldots \nu\ell_n\ \ell_1 := C_1^\bullet;\ \ldots; \ell_n := C_n^\bullet;\ E^\bullet[[\cdot]_i^\bullet(C^\bullet), -]$$

where r is the collection of $\ell_i \mapsto C_i$, and $D^\bullet$ is the context D, in which:

1. the holes $[\cdot]_1$ (to be filled with values V and W) are filled with x.
2. the holes corresponding to $\mathsf{getset}(r)$ are already filled with the corresponding get_{ℓ_i}'s and set_{ℓ_i}'s,

the remaining holes are to be filled with Γ (and *not* Γ'') in (5.11).

□

Note that substitutivity of bisimilarity is restricted either to values ($\mathcal{C}$), or to evaluation contexts ($\mathcal{C}_\mathsf{e}$). Lemma 5.4.6 provides a sufficient condition for a given law between arbitrary terms to be preserved by arbitrary contexts. We first establish weaker results, useful to complete the proof of Lemma 5.4.6.

In the following, we use $\asymp$ to denote any of the relations $\sim, \approx$, and $\gtrsim$.

Lemma 5.4.4. *Suppose that for all s and Γ, we have $(s; \Gamma, L) \asymp (s; \Gamma, R)$. Then for all s, Γ and evaluation context F which might contain free locations, $(s; \Gamma, F[L]) \asymp (s; \Gamma, F[R])$.*

(F might contain free locations, contrarily to above in descriptions of transitions and up to context functions, where contexts are restricted to have no free locations.)

Proof. Let A be the list of set_ℓ and get_ℓ for all location ℓ in F. Then it is easy to get F' from F such that $F = F'[-, A]$ and F' is location-free. By hypothesis we know $(s; \Gamma, A, L) \asymp (s; \Gamma, A, R)$ on which we apply substitutivity for evaluation contexts $\mathcal{C}_\mathsf{e}$ to get $(s; \Gamma, A, F'[L, A]) \asymp (s; \Gamma, A, F'[R, A])$ (Lemmas 5.4.2 and 5.1.9). By weakening w we get $(s; \Gamma, F'[L, A]) \asymp (s; \Gamma, F'[R, A])$. □

Lemma 5.4.5. *Let L, R be ΛR terms with $(s; \Gamma, L) \asymp (s; \Gamma, R)$ for all environment Γ and store s. Suppose C is a multi hole context, such that no hole is in evaluation position in C. Then for all Γ and s we have $(s; \Gamma, C[L]) \asymp (s; \Gamma, C[R])$.*

Proof. We do the proof for the most interesting case, $\asymp = \gtrsim$, and we discuss the other cases at the end of the proof.

Let $\mathcal{R}$ relate each configuration $((\ell \mapsto C_v^\ell[L])_\ell; \tilde{C}_v[L], C[L])$ to the one where R replaces L: $((\ell \mapsto C_v^\ell[R])_\ell; \tilde{C}_v[R], C[R])$ for all $(C_v^\ell)_\ell$ and $\tilde{C}_v$ families of value contexts (of the form $\lambda x.C'$), and where C ranges over contexts where no hole occur in evaluation position. For short we write s_L, s_R, Γ_L, and Γ_R for the corresponding stores and environments. We run simultaneously the transitions from both sides $(s_L; \Gamma_L, C[L])$ and $(s_R; \Gamma_R, C[R])$ as they keep having the same shape.

We prove $\mathcal{R}$ is an expansion relation up to expansion (i.e. $\mathcal{R} \overset{e}{\rightarrowtail} (\gtrsim \mathcal{R})$). We rely on the fact that L and R will never be run directly in this proof.

1. **Silent action** Since in L in $C[L]$ (and R in $C[R]$) is not in evaluation position, both sides will do the same kind of transition, being completely oblivious to L/R. The resulting configurations will be $(s'_L; \Gamma_L, C'[L])$ and $(s'_R; \Gamma_R, C'[R])$. (Even if a set_ℓ or a get_ℓ is involved, and some L/R come from or go to the store, the configurations keep the same shape.)

The only part of the invariant of the relation that is not preserved is that L/R may appear in evaluation position, if $C'[L] = E[L, L]$ (where $[\cdot]_1$ is in evaluation position and $[\cdot]_2$ may appear everywhere).

In this case, we remark that $F \triangleq E[-, L]$ is an evaluation context, on which we can apply Lemma 5.4.4 to have $(s'_L; \Gamma_L, E[L, L]) \gtrsim (s'_L; \Gamma_L, E[R, L])$. Now, since R is not in evaluation position, the context $C_2 = E[R, -]$ is a context with no hole in evaluation position. Hence $(s'_L; \Gamma_L, E[R, L])\ \mathcal{R}\ (s'_R; \Gamma_R, E[R, R])$ and we have closed the diagram.

Note that it may happen that even if $E[L, -]$ does not have holes in evaluation position, $E[R, -]$ does. In this case, we just use Lemma 5.4.4 to transform L's into R's one at at time, as long as there are still such holes in evaluation position. The progression to $\gtrsim\mathcal{R}$ still holds, since $\gtrsim$ is transitive.

2. **Visible action.** First, $C[L]$ is a value iff $C[R]$ is a value so they have the same visible actions of the form $i, D, \mathtt{cod}(r)$. We end up in the same shape of configurations we had for the τ transition above, and proceed the same to close the diagram.

Finally we have proved that $\mathcal{R}$ progresses to $\gtrsim\mathcal{R}$ (expansion up to expansion). In the strong case, we need to prove that $\mathcal{R}$ progresses to $\sim\mathcal{R}$, and in the weak case we need that $\mathcal{R}$ weakly progresses to both $\approx\mathcal{R}$ *and* $\mathcal{R}\approx$, which is necessary because in the weak case, one can use "up to $\approx$" only when $\approx$ is not on the same side as the challenge. □

Lemma 5.4.6. *Let $\asymp$ be any of the relations $\sim, \approx$, and $\gtrsim$. Suppose L, R are ΛR terms with $(s; \Gamma, L) \asymp (s; \Gamma, R)$ for all environments Γ and store s. Then also $(s; \Gamma, C[L]) \asymp (s; \Gamma, C[R])$, for any store s, environment Γ and context C.*

Proof. Using Lemma 5.4.4 and transitivity of $\asymp$, we rewrite first L into R as long as there is a L in evaluation position, and repeat until there is no such L (which can happen several times if L is not a value but R is, for example if $L = II$ and $R = I$, then L is in evaluation position in LL on the right and in LR on the left). We then apply Lemma 5.4.5. □

To get Lemma 5.4.6, we combine the results for evaluation contexts (Lemma 5.4.4) and for non-evaluation contexts (Lemma 5.4.5). This separation is critical, as handling all contexts together would yield a much bigger and error-prone bisimulation candidate as L and R in Lemma 5.4.5 would be replaced by all intricate combinations of E and E'.

In the following $\mathcal{R}^+$ is the transitive closure of $\mathcal{R}$.

Lemma 5.4.7. *Suppose that E and E' are evaluation contexts and that for all value V and store s, we have $(s; E[V]) \longmapsto_{\mathsf{R}}^{+} (s; E'[V])$. Then for all environment Γ and store s, we have $(s; \Gamma, E[M]) \gtrsim (s; \Gamma, E'[M])$.*

Proof. For a given Γ we consider $\mathcal{R} = \{(s; \Gamma, E[M]), (s; \Gamma, E'[M]) \mid \text{for any } s \text{ and } M\}$ and the transitions from both sides:

1. when M is not a value, $(s; M) \longmapsto_{\mathsf{R}} (s'; M')$ and the only transition from both sides is a τ staying in the relation.
2. (left to right) when $M = V$ by hypothesis $(s; \Gamma, E[V]) \xrightarrow{\tau}{}^{+} (s; \Gamma, E'[V])$ so the first transition from the left-hand side is a τ. We use up to expansion to reach $(s; \Gamma, E'[V])$ which is equal to the right-hand side, and conclude up to reflexivity.
3. (right to left) when $M = V$ and the right-hand side makes some transition $(s; \Gamma, E'[V]) \xrightarrow{\alpha} (s'; \Gamma', N')$ we know in fact that $(s; \Gamma, E[V]) \xrightarrow{\tau}{}^{+} (s; \Gamma, E'[V])$ so $(s; \Gamma, E'[V]) \Longrightarrow\!\!\xrightarrow{\alpha} (s'; \Gamma', N')$ and we conclude again up to reflexivity.

We proved $\mathcal{R}$ is an expansion relation up to expansion and reflexivity. □

Corollary 5.4.8. *Suppose that E and E' are evaluation contexts and that for all value V value and store s, we have $(s; E[V]) \longmapsto_{\mathsf{R}}^{*} (s; E'[V])$. Then for all environment s and context C, we have $(s; C[E[M]]) \gtrsim (s; C[E'[M]])$.*

Proof. Consequence of Lemma 5.4.7 and Lemma 5.4.6. □

We use this lemma at various places in the example we present in Section 5.4.1. For instance we use it to replace a term $N_1 \triangleq (\lambda x.E[x])M$ (with E an evaluation context) with $N_2 \triangleq E[M]$, under an arbitrary context. Such a property is delicate to prove, even for closed terms, because the evaluation of M could involve reading from a location of the store that itself could contain occurrences of N_1 and N_2.

5.4.1 An example

We discuss an example from [KW06]. It consists in proving a law between terms of ΛR extended with integers, operators for integer addition and subtraction, and a conditional—those constructs are straightforward to accommodate in the presented framework. For readability, we also use the standard notation for store assignment, dereference and sequence: $(\ell := M) \triangleq \mathsf{set}_\ell M$, $!\ell \triangleq \mathsf{get}_\ell I$, and $M; N \triangleq (\lambda x.N)M$ where x does not appear in N. The two terms are the following ones:

- $M \triangleq \lambda g.\nu \ell\ \ell := 0; g(\mathsf{incr}_\ell); \mathtt{if}\ !\ell\ \mathtt{mod}\ 2 = 0\ \mathtt{then}\ I\ \mathtt{else}\ \Omega$
- $N \triangleq \lambda g.g(F); I$,

where $\mathsf{incr}_\ell \triangleq \lambda z.\ell := !\ell + 2$, and $F \triangleq \lambda z.I$. Intuitively, those two terms are weakly bisimilar because the location bound by ℓ in the first term will always contain an even number.

This example is also considered in [SKS11] where it is however modified to fit the up-to techniques considered in that paper. The latter are less powerful than those available here thanks to the theory of up-to techniques for first-order LTSes (e.g., up to expansion is not considered in [SKS11]—its addition to environmental bisimulations is non-trivial, because one needs to handle stores and environments).

We consider two proofs of the example. In comparison with the proof in [SKS11]: (i) we handle the original example from [KW06], and (ii) the availability of a broader set of up-to techniques and the possibility of freely combining them allows us to work with smaller relations. In the first proof we work up to the store (through the function store) and up to expansion—two techniques that are not available in [SKS11]. In the second proof we exploit the up to transitivity technique of Section 5.1, which is only sound for strong bisimilarity, to further reduce the size of the relation we work with.

First proof. We first employ Corollary 5.4.8 to reach a variant similar to that of [SKS11]: we make a 'thunk' out of the test in M, and we make N look similar. More precisely, let $\mathsf{test}_\ell \triangleq \lambda z.\mathtt{if}\ !\ell\ \mathtt{mod}\ 2 = 0\ \mathtt{then}\ I\ \mathtt{else}\ \Omega$, we first prove that

- $M \approx M' \triangleq \lambda g.\nu \ell\ \ell := 0; g(\mathsf{incr}_\ell); \mathsf{test}_\ell I$, and
- $N \approx N' \triangleq \lambda g.g(F); FI$.

It then suffices to prove that $M' \approx N'$, which we do using the following relation:

$$\mathcal{R} \triangleq \left\{ \left(s, M', (\mathsf{incr}_\ell, \mathsf{test}_\ell)_{\ell \in \tilde{\ell}}\right), \left(\emptyset, N', (F, F)_{\ell \in \tilde{\ell}}\right) \ \text{ s.t. } \forall \ell \in \tilde{\ell},\ \ s(\ell) \text{ is even} \right\} .$$

In the definition of $\mathcal{R}$, the initial pair of terms is generalised by adding any number of private locations, since M' can use itself to create more of them. We prove that relation $\mathcal{R}$ is a weak bisimulation up to store, $\mathcal{C}$ and expansion.

We write $(s; \Gamma_{\tilde{\ell}})$ for the left-hand side of a pair in $\mathcal{R}$ and $(\emptyset; \Delta_{\tilde{\ell}})$ for the right-hand side. Consider a transition $1, C, \mathtt{cod}(r)$ from M' and N'. We write below Γ' for $\Gamma_{\tilde{\ell}}, \mathsf{getset}(r)$ and Δ' for $\Delta_{\tilde{\ell}}, \mathsf{getset}(r)$.

- $(s; \Gamma_{\tilde{\ell}}) \xrightarrow{1, C, \mathtt{cod}(r)} (s \uplus r[\Gamma']; \Gamma', \nu \ell\ \ell := 0; C[\Gamma'](\mathsf{incr}_\ell); \mathsf{test}_\ell I)$
- $(\emptyset; \Delta_{\tilde{\ell}}) \xrightarrow{1, C, \mathtt{cod}(r)} (r[\Delta']; \Delta', C[\Delta'](F); FI)$

In the first line, we make the configuration run two τ transitions, so that $\nu\ell$ and $\ell := 0$ get executed. Now we have a new store $s' = s \uplus (\ell \mapsto 0)$ ($s'(\ell)$ is even, so in this respect we stay in the bisimulation candidate).

Now the main term is $C[\Gamma'](\mathsf{incr}_\ell); \mathsf{test}_\ell I)$ which can be rewritten to $D[\Gamma_{\tilde{\ell}.\ell}, \mathsf{getset}(r)]$ for some context D. On the right-hand side $C[\Delta'](F); FI$ can be rewritten to $D[\Delta_{\tilde{\ell}.\ell}, \mathsf{getset}(r)]$ as well. By construction $(s'; \Gamma_{\tilde{\ell}.\ell})\ \mathcal{R}\ (\emptyset; \Delta_{\tilde{\ell}.\ell})$ hence

$$(s' \uplus r[\Gamma']; \Gamma_{\tilde{\ell}.\ell}, \mathsf{getset}(r)) \quad \mathsf{store}(\mathcal{R}) \quad (r[\Delta']; \Delta_{\tilde{\ell}.\ell}, \mathsf{getset}(r))\ .$$

Now we apply $\mathcal{C}$ with the context D, then the weakening w to remove incr_ℓ and test_ℓ to reach the pair we wanted.

Now that we handled M' and N', let us look at any transition $i, C, \mathsf{cod}(r)$ coming from some incr_ℓ (and F on the other side). It will result in I on both sides (the argument C is discarded), with $s(\ell)$ being updated to $s(\ell) + 2$ which stays in the relation. The store is augmented with $r[\Gamma']$ and $r[\Delta']$ and the environment with $\mathsf{getset}(r)$ which can be safely removed using the store technique as we did before. The same is done when a test_ℓ is run: both sides reduce to I, the argument is discarded, and the r part of the transition is garbage-collected.

Second proof. Here we also preprocess the terms using Corollary 5.4.8 to add a few artificial internal steps to N, so that we can carry out the remainder of the proof using strong bisimilarity, which enjoys more up-to techniques than weak bisimilarity:

- $M \approx M' \triangleq \lambda g.\nu\ell\ \ell := 0; g(\mathsf{incr}_\ell); \mathsf{test}_\ell I$,
- $N \approx N'' \triangleq \lambda g.I; I; g(\mathsf{incr}_0); \mathsf{test}_0 I$.

where incr_0 and test_0 just return I on any input, taking the same number of internal steps as incr_ℓ and test_ℓ. We show that $M' \sim N''$ by proving that the following relation $\mathcal{S}$ is a strong bisimulation *up to unfolding, store, weakening, strengthening, transitivity and context* (a technique unsound in the weak case):

$$\mathcal{S} \triangleq \{(M', N'')\} \cup \{(\ell \mapsto 2n, \mathsf{incr}_\ell, \mathsf{test}_\ell)\,, (\emptyset, \mathsf{incr}_0, \mathsf{test}_0)\ \text{ s.t. } n \in \mathbb{N}\}\ .$$

This relation uses a single location; there is one pair for each integer that can be stored in the location. In the diagram-chasing arguments for $\mathcal{S}$, essentially a pair of derivatives is proved to be related under the function

$$\mathbf{b}' \circ \mathbf{b}' \circ \mathsf{star} \circ (\mathsf{str} \cup \mathsf{store} \cup \mathcal{C} \cup \mathsf{w})^\omega$$

where $\mathsf{star} : \mathcal{R} \mapsto \mathcal{R}^\star$ is the reflexive-transitive closure function.

This up-to technique, unsound in the weak case (transitivity is unsound), is powerful enough to make the bisimulation considerably smaller. Proving that the second member of $\mathcal{S}$ progresses to itself (up to store) is straightforward. We focus on the following transitions from M' and N'':

$$(\emptyset, M') \xrightarrow{1, C, \mathsf{cod}(r)} (\ r[\Gamma];\ \ \Gamma, \nu\ell\ \ell := 0; C[\Gamma]\,(\mathsf{incr}_\ell); \mathsf{test}_\ell I\ ) \triangleq H_1$$
$$(\emptyset, N'') \xrightarrow{1, C, \mathsf{cod}(r)} (\ r[\Delta];\ \Delta, I; I; \qquad C[\Delta](\mathsf{incr}_0); \mathsf{test}_0 I\ ) \triangleq H_2$$

where $\Gamma = M', \mathsf{getset}(r)$ and $\Delta = N'', \mathsf{getset}(r)$. We use $\mathbf{b}'$ as an up-to technique[3] twice to run two steps of reduction on both sides:

$$H_1 \xrightarrow{\tau}\xrightarrow{\tau} H_1' \qquad \text{and} \qquad H_2 \xrightarrow{\tau}\xrightarrow{\tau} H_2'\ .$$

This way we trigger $\nu\ell$ and $\ell := 0$ and obtain two configurations H_1' and H_2' that can be related using a few up-to functions:

$$\begin{array}{rl} & (\ r[\Gamma] \uplus (\ell \mapsto 0);\ \Gamma,\ C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I\ ) = H_1' \\ \mathsf{w}(\mathcal{C}(\mathsf{store}(\mathsf{str}(\mathcal{S})))) & (\ r[\Gamma];\ \Gamma,\ C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I\ ) \\ \mathcal{C}(\mathsf{store}(\mathcal{S})) & (\ r[\Delta];\ \Delta,\ C[\Delta](\mathsf{incr}_0); \mathsf{test}_0 I\ ) = H_2'\ . \end{array}$$

[3] If $\xrightarrow{\tau}$ is deterministic then $(\xrightarrow{\tau} \mathcal{R} \xleftarrow{\tau}) \subseteq \mathbf{b}'(\mathcal{R})$.

We detail below how we go from the first to the second line. We write $\Gamma_\ell \triangleq \mathsf{incr}_\ell, \mathsf{test}_\ell$ and $\Gamma_0 \triangleq \mathsf{incr}_0, \mathsf{test}_0$.

$$\begin{array}{rcl}
(\ell \mapsto 0; \Gamma_\ell) & \mathcal{S} & (\emptyset; \Gamma_0) \\
(\ell \mapsto 0; \Gamma_\ell, M') & \mathsf{str}(-) & (\emptyset; \Gamma_0, M') \\
(r[\Gamma] \uplus \ell \mapsto 0; \Gamma_\ell, \Gamma) & \mathsf{store}(-) & (r[\Gamma]; \Gamma_0, \Gamma) \\
(r[\Gamma] \uplus \ell \mapsto 0; \Gamma_\ell, \Gamma, C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I)) & \mathcal{C}(-) & (r[\Gamma]; \Gamma_0, \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I) \\
(r[\Gamma] \uplus \ell \mapsto 0; \quad \Gamma, C[\Gamma](\mathsf{incr}_\ell); \mathsf{test}_\ell I)) & \mathsf{w}(-) & (r[\Gamma]; \quad \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I)
\end{array}$$

Going from the second to the third line is easier:

$$\begin{array}{rcl}
(\emptyset; M') & \mathcal{S} & (\emptyset; N'') \\
(r[\Gamma]; \Gamma) & \mathsf{store}(\mathcal{S}) & (r[\Delta]; \Delta) \\
(r[\Gamma]; \Gamma, C[\Gamma](\mathsf{incr}_0); \mathsf{test}_0 I) & \mathcal{C}(\mathsf{store}(\mathcal{S})) & (r[\Delta]; \Delta, C[\Delta](\mathsf{incr}_0); \mathsf{test}_0 I)
\end{array}$$

Finally we proved that $H_1 \; f(\mathcal{S}) \; H_2$ where $f = \mathbf{b}' \circ \mathbf{b}' \circ \mathsf{star} \circ (\mathsf{str} \cup \mathsf{store} \cup \mathcal{C} \cup \mathsf{w})^\omega$ is a compatible function, and hence $\mathcal{S} \rightarrowtail f(\mathcal{S}) \cup \mathsf{store}(\mathcal{S})$ (not forgetting the second member of $\mathcal{S}$).

To conclude, $\mathcal{S}$, as a strong bisimulation up to (unfolding, store, weakening, strengthening, transitivity and context), is included in $\sim$.

About the two proofs The difference between the relation $\mathcal{R}$ in the first proof and the proofs in [KW06, SKS11] is that $\mathcal{R}$ only requires locations that appear free in the tested terms; in contrast, the relations in [KW06, SKS11] need to be closed under all possible extensions of the store, including extensions in which related locations are mapped onto arbitrary context-closures of related values. We avoid this thanks to the up to store function. The reason why, both in [KW06, SKS11] and in the first proof above, several locations have to be considered is that, with bisimulations akin to environmental bisimulation, the input for a function is built using the values that occur in the candidate relation. In our example, this means that the input for a function can be a context-closure of M and N; hence uses of the input may cause several evaluations of M and N, each of which generates a new location. In this respect, it is surprising that our second proof avoids multiple allocations (the candidate relation $\mathcal{S}$ only mentions one location). This is due to the massive combination of up-to techniques whereby, whenever a new location is created, a double application of up to context (the 'double' is obtained from up to transitivity) together with some administrative work (given by the other techniques) allows us to absorb the location.

5.5 Conclusion: applicability of the method

In this section, we assess the relevance of first-order LTSes for a few common calculi, and give an account for each of them to say how the most straightforward approaches would work. We observe, as in Remark 5.3.12 for the lambda-calculus, that full abstraction is in general not sufficient if one desires to have up to context techniques (in fact, in this section, we do not talk about the full abstraction correspondences with the original calculi).

This section focuses on the problems we encounter when studying up to context functions in first-order LTSes, in the hope of providing the reader with a general intuition of what is likely to work, and what is not.

5.5.1 Asynchronous bisimilarities

In asynchronous process calculi, the continuations of output prefixes are always 0. The literature usually studies asynchronous variants of the π-calculus, but in this section we focus on a problem that is not related to name-passing. Therefore, we use instead an asynchronous variant of CCS, written ACCS and defined as the following subcalculus of CCS:

$$P \;\; ::= \;\; 0 \;\mid\; P \mid Q \;\mid\; (\boldsymbol{\nu} a)P \;\mid\; \overline{a} \;\mid\; a.P \;\mid\; P + Q \;\; .$$

To capture barbed congruence (as remarked in Section 2.1.2, there are no input barbs in this calculus since $\overline{a}.\overline{w}$ is not a valid process) for this calculus, one need different notions of bisimilarities [ACS96, HT91]:

Definition 5.5.1. Strong asynchronous bisimilarity is the largest symmetric relation $\sim_{\mathsf{a}}$ such that, whenever $P \sim_{\mathsf{a}} Q$ and $P \stackrel{\mu}{\longrightarrow} P'$, we have:

1. either $Q \stackrel{\mu}{\longrightarrow} Q'$ and $P' \sim_{\mathsf{a}} Q'$,
2. or $\mu = a$ is an input action and $Q \stackrel{\tau}{\longrightarrow} Q'$ with $P' \sim_{\mathsf{a}} (Q' \mid \overline{a})$.

Weak asynchronous bisimilarity $\approx_{\mathsf{a}}$ is defined similarly, replacing $Q \stackrel{\mu}{\longrightarrow} Q'$ with $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ and $Q \stackrel{\tau}{\longrightarrow} Q'$ with $Q \Longrightarrow Q'$.

These asynchronous bisimilarities respectively coincide with strong and weak barbed congruence ($\sim_{\mathsf{a}} = \simeq_{\mathrm{ACCS}}$ and $\approx_{\mathsf{a}} = \cong_{\mathrm{ACCS}}$). Canonical examples of asynchronous equalities are:

$$\tau \sim_{\mathsf{a}} \tau + a.\overline{a} \qquad\qquad 0 \approx_{\mathsf{a}} a.\overline{a} \ . \tag{5.12}$$

These two examples immediately show that the bisimilarities induced by the standard LTS for CCS are not coarse enough, as the right-hand sides can both perform the a input action, whereas the left-hand sides only have silent actions. However, it is possible to define an LTS with the usual processes and labels, with transitions $\stackrel{\cdot}{\longrightarrow}_1$ defined as follows:

$$\frac{P \stackrel{\overline{a}}{\longrightarrow} P'}{P \stackrel{\overline{a}}{\longrightarrow}_1 P'} \qquad\qquad \frac{P \stackrel{\tau}{\longrightarrow} P'}{P \stackrel{\tau}{\longrightarrow}_1 P'} \qquad\qquad \frac{}{P \stackrel{a}{\longrightarrow}_1 P \mid \overline{a}} \ .$$

Note that $\stackrel{a}{\longrightarrow}_1$ does not have the usual meaning, it only ensures that every bisimulation is closed under contexts of the form $[\cdot] \mid \overline{a}$ (this corresponds to the notion of 1-bisimulation in [ACS96]). This LTS induces bisimilarities that coincide with barbed congruence: $\sim_1 = \sim_{\mathsf{a}}$ and $\approx_1 = \approx_{\mathsf{a}}$. The problem is that the usual up to context techniques are unsound in this setting, for example using "up to context $[\cdot] \mid \overline{a}$" would cancel the $\stackrel{a}{\longrightarrow}_1$ transition.

Using the same method as for the lambda-calculus, we can define another LTS with transitions $\stackrel{\cdot}{\longrightarrow}_2$:

$$\frac{P \stackrel{\overline{a}}{\longrightarrow} P'}{P \stackrel{\overline{a}}{\longrightarrow}_2 P'} \qquad\qquad \frac{P \stackrel{\tau}{\longrightarrow} P'}{P \stackrel{\tau}{\longrightarrow}_2 P'} \qquad\qquad \frac{P \mid \overline{a} \stackrel{\tau}{\longrightarrow} P'}{P \stackrel{a}{\longrightarrow}_2 P'} \ .$$

The corresponding strong bisimilarity coincides with asynchronous strong bisimilarity: $\sim_2 = \sim_{\mathsf{a}}$ and we have now that the usual up to context techniques are sound (for example, up to parallel composition). The problem is that now, $\approx_2 \neq \approx_{\mathsf{a}}$ since in (5.12) process 0 has no $\stackrel{a}{\longrightarrow}_2$ transition.

To capture $\approx_{\mathsf{a}}$, one can define yet another LTS $\stackrel{\cdot}{\longrightarrow}_3$:

$$\frac{P \stackrel{\overline{a}}{\longrightarrow} P'}{P \stackrel{\overline{a}}{\longrightarrow}_3 P'} \qquad\qquad \frac{P \stackrel{\tau}{\longrightarrow} P'}{P \stackrel{\tau}{\longrightarrow}_3 P'} \qquad\qquad \frac{P \mid \overline{a} \stackrel{\tau}{\longrightarrow}^{=} P'}{P \stackrel{a}{\longrightarrow}_3 P'} \ ,$$

where $\stackrel{\tau}{\longrightarrow}^{=}$ is the reflexive closure of $\stackrel{\tau}{\longrightarrow}$. Note that $\stackrel{\cdot}{\longrightarrow}_3$ is the union of $\stackrel{\cdot}{\longrightarrow}_1$ and $\stackrel{\cdot}{\longrightarrow}_2$. We have now that $\approx_3 = \approx_{\mathsf{a}}$ but $\sim_3 \neq \sim_{\mathsf{a}}$ (since $0 \sim_3 a.\overline{a}$). Up to context techniques work for $\sim_3$ and $\approx_3$ as well. In addition, the technique "$\approx_3$-bisimulation up to $\sim_2$" is sound, but needs a separate proof: it is *not* a consequence of a theorem in the general theory of up-to techniques, since the LTSes are different.

The conclusion of this is that there seems to be no simple first-order LTS (with *Pr* being the regular set of processes) for ACCS admitting up to context techniques and capturing both $\sim_{\mathsf{a}}$ and $\approx_{\mathsf{a}}$.

5.5.2 Higher-order π-calculus

In the higher-order π-calculus, communicated data can include processes themselves. We consider the calculus HOπ described by the following grammar:

$$P \ ::= \ 0 \ \mid \ P \mid Q \ \mid \ (\boldsymbol{\nu} a)P \ \mid \ \overline{a}\langle P \rangle.Q \ \mid \ a(X).P \ \mid \ X$$

with the following reduction rule:

$$\overline{a}\langle R\rangle.P \mid a(X).Q \longrightarrow P \mid Q\{R/X\}$$

which we close under usual structural congruence, restriction and parallel contexts, and where $Q\{R/X\}$ denotes capture-avoiding substitution of processes. Bisimulations for higher-order π-calculus are not straightforward. The literature proposes several notions of bisimulations, and we study here those that seem most suited for establishing first-order LTSes. We reason about closed processes, i.e. processes with no free process variables.

Context bisimulation [San96a] The context bisimulation corresponds to the following LTS:

$$\mathit{Act} : \mu \quad ::= \quad \tau \;\mid\; aP \;\mid\; \overline{a}C \qquad \frac{P \xrightarrow{\tau}_{\mathsf{HO}\pi} P'}{P \xrightarrow{\tau} P'} \qquad \frac{P \xrightarrow{aR}_{\mathsf{HO}\pi} P'}{P \xrightarrow{aR} P'} \qquad \frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{a}\langle R\rangle}_{\mathsf{HO}\pi} P'}{P \xrightarrow{\overline{a}C} (\boldsymbol{\nu}\tilde{c})(P' \mid C[R])}$$

where $\xrightarrow{\cdot}_{\mathsf{HO}\pi}$ is defined in an early style on $\mathsf{HO}\pi$ processes, and C ranges over multi-hole contexts. The idea is that each time a process R is emitted, an external process could add any process of the form $C[R]$ in parallel. Up to context techniques accommodate well in this setting, with verbose but rather straightforward compatibility proofs. These proofs are rather easy, which mainly comes from the fact that the output transition already closes bisimulations under contexts built from emitted processes.

Lemma 5.5.2. *The following techniques are compatible up to:*

$$\frac{P \;\mathcal{R}\; Q}{(\boldsymbol{\nu}a)P \;\; \mathcal{C}_\nu(\mathcal{R}) \;\; (\boldsymbol{\nu}a)Q} \qquad \frac{\forall R \;\; P\{R/X\} \;\mathcal{R}\; Q\{R/X\}}{a(X).P \;\; \mathcal{C}_i(\mathcal{R}) \;\; a(X).Q} \qquad \frac{P \;\mathcal{R}\; Q}{\overline{a}\langle R\rangle.P \;\; \mathcal{C}_o(\mathcal{R}) \;\; \overline{a}\langle R\rangle.Q} \qquad \frac{P \;\mathcal{R}\; Q}{P \mid R \;\; \mathcal{C}_|(\mathcal{R}) \;\; Q \mid R}$$

Proof. Functions $\mathcal{C}_0$ and $\mathcal{C}_\nu$ are compatible up to identity, and $\mathcal{C}_o$ is compatible up to $\mathcal{C}_|$. For $\mathcal{C}_|$ itself, observe that the non-trivial transitions from $P \mid R$ are silent transitions to $(\boldsymbol{\nu}\tilde{c})(P' \mid R')$ where:

1. either $P \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{a}\langle S\rangle}_{\mathsf{HO}\pi} P'$ and $R \xrightarrow{aS}_{\mathsf{HO}\pi} R'$ with $R' = C[S]$ for some C. This corresponds exactly to a first order transition $\overline{a}C$ from P and we simply progress to $\mathcal{S}$.
2. or $R \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{a}\langle S\rangle}_{\mathsf{HO}\pi} R'$ with $P \xrightarrow{aS}_{\mathsf{HO}\pi} P'$ which corresponds again to the first order transition aS from P, and we progress to $\mathcal{C}_|(\mathcal{S})$.

In both case, we derive straightforwardly the same transition from the RHS Q. □

Environmental bisimulation [SKS11] Another relevant notion is environmental bisimulation from [SKS11], which corresponds to the following LTS, where environments record the emitted processes:

$$\mathit{Pr} = \{(\tilde{a}, \Gamma, P) \;\mid\; \tilde{a} \text{ is a set of names, } \Gamma \text{ is a sequence of closed processes, } P \text{ is closed}\}$$

$$\mathit{Act} : \mu \quad ::= \quad \tau \;\mid\; aC \;\mid\; \overline{a} \;\mid\; i \qquad (i \in \mathbb{N})$$

$$\frac{P \xrightarrow{\tau}_{\mathsf{HO}\pi} P'}{(\tilde{a}, \Gamma, P) \xrightarrow{\tau} (\tilde{a}, \Gamma, P')} \qquad \frac{P \xrightarrow{bR}_{\mathsf{HO}\pi} P' \qquad R = C[\Gamma] \qquad b \notin \tilde{a} \qquad \mathrm{n}(C) \cap \tilde{a} = \emptyset}{(\tilde{a}, \Gamma, P) \xrightarrow{bC} (\tilde{a}, \Gamma, P')}$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{b}\langle R\rangle}_{\mathsf{HO}\pi} P' \qquad b \notin \tilde{a}\tilde{c}}{(\tilde{a}, \Gamma, P) \xrightarrow{\overline{b}} (\tilde{a}\tilde{c}, \Gamma, R, P')} \qquad \frac{}{(\tilde{a}, \Gamma, P) \xrightarrow{i} (\tilde{a}, \Gamma, P \mid \Gamma_i)}$$

We write $\mathrm{n}(C)$ for $\mathrm{fn}(C[0]) \cup \mathsf{cn}(C)$ where $\mathsf{cn}(C)$ is the set of names captured by C.

The role of the i action is to put back emitted processes R in the tested process P at any point, closing bisimulations under parallel compositions of Γ_i (note that the context bisimulation is less subtle, as it imposes to close under parallel compositions of $C[\Gamma]$).

Again we obtain full abstraction, but we get again a problem with up to context, as the transition i can be cancelled by context[4] $C_i = [\cdot] \mid [\cdot]_i$:

$$x \triangleq (\tilde{a}, \Gamma, P) \xrightarrow{i} (\tilde{a}, \Gamma, P \mid \Gamma_i) = C_i[x] \ . \tag{5.13}$$

(In fact $x \xrightarrow{i} x'$ implies $x' = C_i[x]$ for all x and x'.) One can fix this using a trick similar to the one for the lambda-calculus, by adding transitions of the form:

$$\frac{\Gamma_I \mid P \xrightarrow{\tau}_{\mathsf{HO}\pi} P \qquad |I| \leq 2}{(\tilde{a}, \Gamma, P) \xrightarrow{I,\tau} (\tilde{a}, \Gamma, P')} \text{ where } \Gamma_I = \Gamma_{i_1} \mid \ldots \mid \Gamma_{i_n}$$

where we require that I is a sequence of integers (of length at most 2). If we transport the same kind of transitions to the other labels, we can have forms of up-to techniques together with full abstraction, but only in the strong case. Unfortunately, we have found no simple way to correct this approach to work for weak bisimilarity even when resorting to a non-uniform treatment of strong and weak bisimilarities like we did in Section 5.5.1.

Environmental bisimulation [KH12] Koutavas and Hennessy presented in [KH12] a first-order LTS with environments (which they call knowledge environments), as above, such that bisimulation captures the desired equivalence, very much like the approach we have throughout this chapter, but without providing up to context proof techniques. The calculus they use is rather general and slightly different in several ways. In particular, we focus on their notion of value: in their setting, a value is a "thunked process", written $\lambda.P$, that represents a process P waiting to be "run". We import this precise ingredient from [KH12] and otherwise stick to the same setting as above, to facilitate comparisons:

$$\begin{array}{rcl} P & ::= & 0 \mid P \mid Q \mid (\boldsymbol{\nu} a)P \mid \overline{a}\langle V\rangle.Q \mid a(X).P \mid \mathsf{app}V \\ V & ::= & X \mid \lambda.P \end{array}$$

There are now two reduction rules:

$$\overline{a}\langle V\rangle.P \mid a(X).Q \longrightarrow P \mid Q\{V/X\} \qquad\qquad \mathsf{app}(\lambda.P) \longrightarrow P \ .$$

We define the following LTS:

$$\mathit{Pr} = \{(\tilde{a}, \Gamma, P) \mid \tilde{a} \text{ is a set of names, } \Gamma \text{ is a sequence of closed values, } P \text{ is closed}\}$$

$$\mathit{Act} : \mu \ ::= \ \tau \mid aC_v \mid \overline{a} \mid i$$

$$\frac{P \xrightarrow{\tau}_{\mathsf{HO}\pi} P'}{(\tilde{a}, \Gamma, P) \xrightarrow{\tau} (\tilde{a}, \Gamma, P')} \qquad \frac{P \xrightarrow{bV}_{\mathsf{HO}\pi} P' \qquad V = C_v[\Gamma] \qquad b \notin \tilde{a} \qquad \mathrm{n}(C_v) \cap \tilde{a} = \emptyset}{(\tilde{a}, \Gamma, P) \xrightarrow{bC_v} (\tilde{a}, \Gamma, P')}$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{b}\langle V\rangle}_{\mathsf{HO}\pi} P' \qquad b \notin \tilde{a}\tilde{c}}{(\tilde{a}, \Gamma, P) \xrightarrow{\overline{b}} (\tilde{a}\tilde{c}, \Gamma, V, P')} \qquad \frac{\Gamma_i = \lambda.R}{(\tilde{a}, \Gamma, P) \xrightarrow{i} (\tilde{a}, \Gamma, P \mid R)}$$

[4] One could object that we should first try simpler contexts that do not use pieces of context from the environment Γ (which is the role of $[\cdot]_i$ in C_i). However, the compatibility of such simpler contexts demands the compatibility of the more involved contexts. For example, the simple context $[\cdot] \mid a(X).C[X]$ will add $C[\Gamma]$ in parallel of the tested process.

This time, C_v is a value context of the form $\lambda.C$ where C is a regular context.

The only difference is the action i, which triggers the thunked content of the environment value Γ_i before adding it in parallel of processes. This small difference makes the usual up to context problem disappear, as now $P \mid R$ cannot be built as a context of P and $\Gamma_i = \lambda.R$. Importantly, in [KH12], the corresponding rule creates the process $P \mid \mathsf{app}(\lambda.R)$ which *can* be built as a context of P and $\lambda.R$, as in (5.13), forbidding yet again any up to context.

Note that, when reasoning with environments, bisimilarity is not closed under arbitrary contexts. For example, using a typical law in presence of replication: since $\overline{a}\langle\overline{c}\rangle.!\overline{c} \simeq_{\mathsf{HO}\pi} \overline{a}\langle 0\rangle.!\overline{c}$, we have also $(\lambda.\overline{c}, !\overline{c}) \sim (\lambda.0, !\overline{c})$. However, $(\lambda.\overline{c}, \overline{b}.!\overline{c}) \not\sim (\lambda.0, \overline{b}.!\overline{c})$ so the context $\overline{b}.[\cdot]$ does not preserve bisimilarity (we write $\overline{c}$ for $\overline{c}\langle 0\rangle.0$ and $\overline{b}.$ for $\overline{b}\langle 0\rangle.$). For this reason, the method of initial contexts does not work here, and we get techniques close to those for the lambda-calculus.

Lemma 5.5.3. *The following techniques are compatible up to $\sim$:*

$$\frac{(\tilde{a}, \Gamma, V, P)\ \ \mathcal{R}\ \ (\tilde{b}, \Delta, W, Q)}{(\tilde{a}, \Gamma, P)\ \ \mathsf{w}(\mathcal{R})\ \ (\tilde{b}, \Delta, Q)} \qquad \frac{(\tilde{a}, \Gamma, P)\ \ \mathcal{R}\ \ (\tilde{b}, \Delta, Q)}{(\tilde{a}, \Gamma, C_v[\Gamma], P)\ \ \mathsf{str}(\mathcal{R})\ \ (\tilde{b}, \Delta, C_v[\Gamma], Q)}$$

$$\frac{(\tilde{a}, \Gamma, P)\ \ \mathcal{R}\ \ (\tilde{b}, \Delta, Q) \qquad \mathrm{n}(C) \cap (\tilde{a} \cup \tilde{b}) = \emptyset}{(\tilde{a}, \Gamma, P \mid C[\Gamma])\ \ \mathcal{C}_|(\mathcal{R})\ \ (\tilde{b}, \Delta, Q \mid C[\Delta])} \qquad \frac{(\tilde{a}, \Gamma, P)\ \ \mathcal{R}\ \ (\tilde{b}, \Delta, Q) \qquad \tilde{c} \cap (\tilde{a} \cup \tilde{b}) = \emptyset}{(\tilde{c} \cup \tilde{a}, \Gamma, P)\ \ \mathcal{C}_\nu(\mathcal{R})\ \ (\tilde{c} \cup \tilde{b}, \Delta, Q)}$$

Proof. We consider proofs for **b** and **wb** at the same time, since there is no important difference. Compatibilities of w and $\mathcal{C}_\nu$ are straightforward. Then, remark that

$$(\{b\} \cup \tilde{a}, \Gamma, P) \sim (\tilde{a}, \Gamma, (\boldsymbol{\nu} b)P) \tag{5.14}$$

when $b \notin \mathrm{fn}(\Gamma) \cup \tilde{a}$, which means that the compatibility of $\mathcal{C}_\nu$ and $\mathcal{R} \mapsto \sim\mathcal{R}\sim$ implies the compatibility of the technique "up to restriction", with contexts of the form $(\boldsymbol{\nu} b)[\cdot]$ when b is not used in the environment. Function str progresses to $\mathsf{str} \cup \mathcal{C}_|$ (the $\mathcal{C}_|$ part is useful for the transitions labelled with aC_v or i).

We now move to the compatibility of $\mathcal{C}_|$ which is the critical part of the proof for up to context. Suppose that $\mathcal{R} \subseteq \mathcal{S}$, that $\mathcal{R}$ progresses to $\mathcal{S}$, and that $(\tilde{a}, \Gamma, P)\ \ \mathcal{R}\ \ (\tilde{b}, \Delta, Q)$ with $\mathrm{n}(C) \cap (\tilde{a} \cup \tilde{b}) = \emptyset$, and consider the transitions from $(\tilde{a}, \Gamma, P \mid C[\Gamma])$. Actions from P alone are easy to handle. Actions from $C[\Gamma]$ alone can be:

1. if $C[\Gamma] \longrightarrow R \mid C'[\Gamma]$ with $\Gamma_i = \lambda.R$ and $C \equiv \mathsf{app}([\cdot]_i) \mid C'$ we use a i transition from P and Q to progress to $\mathcal{C}_|(\mathcal{S})$. This case is particularly interesting, as it is the one that fails for the LTS derived from [SKS11].

2. if $C[\Gamma] \xrightarrow{aV}_{\mathsf{HO}\pi} P_1$ with $V = C_v[\Gamma]$, in which case $C[\Gamma] \xrightarrow{aC_v[\Delta]}_{\mathsf{HO}\pi} Q_1$ with $P_1 = C_1[\Gamma]$ and $Q_1 = C_1[\Delta]$, to progress to $\mathcal{C}_|(\mathcal{R}) \subseteq \mathcal{C}_|(\mathcal{S})$.

3. if $C[\Gamma] \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{b}\langle V\rangle}_{\mathsf{HO}\pi} P_1$ then $V = C_v[\Gamma]$ and $P_1 = C_1[\Gamma]$. We obtain a similar transition from $C[\Delta]$, and we need to relate $(\tilde{c}\tilde{a}, \Gamma, C_v[\Gamma], P \mid C_1[\Gamma])$ to $(\tilde{c}\tilde{b}, \Delta, C_v[\Delta], Q \mid C_1[\Delta])$, which we do through the relation $\mathcal{C}_\nu^\omega(\mathsf{str}(\mathcal{C}_|(\mathcal{R})))$.

We now need to handle the τ transitions that are obtained from a combination of P and $C[\Gamma]$, i.e. $(\tilde{a}, \Gamma, P \mid C[\Gamma]) \xrightarrow{\tau} (\tilde{a}, \Gamma, (\boldsymbol{\nu}\tilde{c})(P' \mid C'[\Gamma]))$. There are two cases (we recall that since the names of C do not intersect with $\tilde{a}$, the communication happens on a public channel x):

1. $P \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{x}\langle V\rangle}_{\mathsf{HO}\pi} P'$ and $C[\Gamma] \xrightarrow{xV}_{\mathsf{HO}\pi} P_1$. By analysing the latter transition, we know that $P_1 = C_1[\Gamma, V]$ for some C_1. Using the fact that $\mathcal{R}$ progresses to $\mathcal{S}$ and that $(\tilde{a}, \Gamma, P) \xrightarrow{\overline{x}} (\tilde{a}\tilde{c}, \Gamma, V, P')$, we know $(\tilde{b}, \Delta, Q) \stackrel{\overline{x}}{\Longrightarrow} (\tilde{b}\tilde{d}, \Delta, W, Q')$ with $Q \xrightarrow{(\boldsymbol{\nu}\tilde{d})\overline{x}\langle W\rangle}_{\mathsf{HO}\pi} Q'$ and $C[\Delta] \xrightarrow{xW}_{\mathsf{HO}\pi} C_1[\Delta, W]$. We relate then the

final configurations using several techniques:

$$\dfrac{\dfrac{\dfrac{(\tilde{a}\tilde{c},\Gamma,V,P')\ \ \mathcal{S}\ \ (\tilde{b}\tilde{d},\Delta,W,Q')}{(\tilde{a}\tilde{c},\Gamma,V,P' \mid C_1[\Gamma,V])\ \ \mathcal{C}_{|}(\mathcal{S})\ \ (\tilde{b}\tilde{d},\Delta,W,Q' \mid C_1[\Delta,W])}}{(\tilde{a}\tilde{c},\Gamma,P' \mid C_1[\Gamma,V])\ \ \mathsf{w}(\mathcal{C}_{|}(\mathcal{S}))\ \ (\tilde{b}\tilde{d},\Delta,Q' \mid C_1[\Delta,W])}}{(\tilde{a},\Gamma,(\boldsymbol{\nu}\tilde{c})(P' \mid C_1[\Gamma,V]))\ \ {\sim}\mathsf{w}(\mathcal{C}_{|}(\mathcal{S})){\sim}\ \ (\tilde{b},\Delta,(\boldsymbol{\nu}\tilde{d})(Q' \mid C_1[\Delta,W]))}$$

using (5.14) (beginning of this proof) to move $\tilde{c}$ inside the tested process.

2. $P \xrightarrow{xV}_{\mathsf{HO}\pi} P'$ and $C[\Gamma] \xrightarrow{(\boldsymbol{\nu}\tilde{c})\overline{x}\langle C_v[\Gamma]\rangle}_{\mathsf{HO}\pi} C_1[\Gamma]$ (every output transition from $C[\Gamma]$ has to be of this shape and hence $V = C_v[\Gamma]$). We derive then the transition xC_v from $(\tilde{a},\Gamma,P)$ to $(\tilde{a},\Gamma,P')$ and similarly for $(\tilde{b},\Delta,Q)$. Finally, we relate $(\tilde{a},\Gamma,(\boldsymbol{\nu}\tilde{c})(P' \mid C_1[\Gamma]))$ to $(\tilde{b},\Delta,(\boldsymbol{\nu}\tilde{c})(Q' \mid C_1[\Delta]))$ through the relation ${\sim}\mathcal{C}_{\nu}^{|\tilde{c}|}(\mathcal{C}_{|}(\mathcal{S})){\sim}$, again using (5.14)

All together, we proved that $\mathcal{C}_{|}$ progresses to $(\mathcal{C}_{|} \cup \mathcal{C}_{\nu}^{\omega} \cup \mathsf{w} \cup \mathsf{str} \cup (\mathcal{R} \mapsto {\sim}\mathcal{R}{\sim}))^{\omega}$ (note that $\mathcal{C}_{\nu}^{\omega}$ is compatible since $\mathcal{C}_{\nu}$ is). □

Note that the first-order LTS of [KH12] has other interesting properties: it handles name passing (being a superset of both HOπ and π) and has the distinctive advantage of dealing with open processes in a nice way, using a symbolic style: transitions on configurations correspond to the following transitions of plain processes:

$$a(X).P \xrightarrow{a(X)} P \qquad\qquad X \xrightarrow{X} 0\ .$$

By contrast, the LTS we use above has a large set of actions: one aC for each context C. We already know that up to context techniques are unsound in the setting of [KH12] for the reason mentioned above. We hope that changing rule CONC-APP-TRANS the same way we used previously $P \mid R$ instead of $P \mid \mathsf{app}(\lambda.R)$ will make the up to context techniques compatible, as it would be the first example of first-order bisimulation with symbolic transitions (the same transitions as CCS with the addition of $\xrightarrow{i}$ transitions) with a complete tool set of up to context proof techniques. We leave this for future work.

5.5.3 Conclusion

Building a first-order labelled transition system that captures the equivalence of interest allows one to fit into the theory of bisimulation enhancements, and inherit all the compatible up-to techniques. However, those up-to techniques are mainly useful when we provide in addition the expected set of compatible up to context techniques. The main difficulty we have faced while establishing the results presented in this chapter was the careful design of labelled transitions, to make the up to context techniques compatible functions through known notions of bisimulations.

In several cases, following the approach involved establishing precise notions of values (e.g. thunks in HOπ and get_{ℓ} and set_{ℓ} in imperative λ) so to have those values treated uniformly, and to satisfy a notion of "consumption" of the value, that cannot be cancelled by up-to techniques.

We have applied this approach to various examples of calculi, systematically highlighting the technical difficulties and overcoming unsuitable design choices. We hope to have thereby helped the reader to a deeper and more uniform understanding of first-order reasoning for higher-order systems.

Bibliography

[Abr89] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

[AC98] R. Amadio and P.-L. Curien. *Domains and Lambda-calculi*. Cambridge University Press, New York, NY, USA, 1998.

[ACCL90] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In Frances E. Allen, editor, *Proc. of POPL*, pages 31–46. ACM Press, 1990.

[ACS96] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. In Ugo Montanari and Vladimiro Sassone, editors, *Proc. of CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 1996.

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL*, pages 104–115. ACM, 2001.

[AG98] M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. In Chris Hankin, editor, *ESOP'98*, volume 1381 of *LNCS*, pages 12–26. Springer, 1998.

[AKH92] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[Bar84] H. P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*, volume 103. North Holland, revised edition, 1984. http://www.cs.ru.nl/ henk/Personal Webpage.

[BBM04] M. Boreale, M. G. Buscemi, and U. Montanari. D-fusion: A distinctive fusion calculus. In Wei-Ngan Chin, editor, *Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 296–310. Springer Berlin Heidelberg, 2004.

[BBM05] M. Boreale, M. G. Buscemi, and U. Montanari. A general name binding mechanism. In *TGC*, volume 3705 of *LNCS*, pages 61–74. Springer, 2005.

[BJPV09] J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *LICS*, pages 39—-48. IEEE, 2009.

[Bor98] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195:205–226, 1998.

[Bou92] G. Boudol. Asynchrony and the pi-calculus. Technical Report 1702, INRIA, 1992.

[Coq] The Coq Proof Assistant. `http://coq.inria.fr`.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984.

[DK04] V. Danos and J. Krivine. Reversible communicating systems. In *Proc. of CONCUR*, pages 292–307, 2004.

[DS06] Y. Deng and D. Sangiorgi. Towards an algebraic theory of typed mobile processes. *Theor. Comput. Sci.*, 350(2-3):188–212, 2006.

[Fu97] Y. Fu. The χ-calculus. In *APDC*, pages 74–81. IEEE Computer Society, 1997.

[Gue14] N. Guenot. Session types, solos, and the computational contents of sequent calculus proofs (abstract), 2014.

[GV89] J.F. Groote and F.W. Vaandrager. Structural operational semantics and bisimulation as a congruence (extended abstract). In *Proc. of ICALP*, pages 423–438, 1989.

[GW00] P. Gardner and L. Wischik. Explicit fusions. In *MFCS*, volume 1893 of *LNCS*, pages 373–382. Springer, 2000.

[GWGJ10] T. Given-Wilson, D. Gorla, and B. Jay. Concurrent pattern calculus. In CristianS. Calude and Vladimiro Sassone, editors, *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer Berlin Heidelberg, 2010.

[HMS12] D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Duality and i/o-types in the π-calculus. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 2012.

[HMS13] D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Name-passing calculi: From fusions to preorders and types. In *LICS*, pages 378–387. IEEE Computer Society, 2013.

[HMX15] D. Hirschkoff, J.M. Madiot, and X. Xu. A behavioural theory for a π-calculus with preorders. In *Proc. of FSEN*, 2015. To appear.

[HR02] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. and Comp.*, 173:82—-120, 2002.

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *ECOOP*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[Hüt13] H. Hüttel. Types for resources in Ψ-calculi. In *Proc. of TGC*, volume 8358 of *Lecture Notes in Computer Science*, pages 83–102, 2013.

[HY95] K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comp. Sci.*, 152(2):437–486, 1995.

[JR99] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *LICS*, pages 56–66, 1999.

[KH12] V. Koutavas and M. Hennessy. First-order reasoning for higher-order concurrency. *Computer Languages, Systems & Structures*, 38(3):242–277, 2012.

[KLS11] V. Koutavas, P. B. Levy, and E. Sumii. From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, 276:215–235, 2011.

[KPT99] N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. *TOPLAS*, 21(5):914–947, 1999. Preliminary summary appeared in Proceedings of POPL'96.

[KW06] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL'06*, pages 141–152. ACM, 2006.

[Las98a] S.B. Lassen. Relational reasoning about contexts. In *Higher-order operational techniques in semantics*, pages 91–135. Cambridge University Press, 1998.

[Las98b] S.B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Department of Computer Science, University of Aarhus, 1998.

[Len98] M. Lenisa. *Themes in Final Semantics*. Ph.D. thesis, Università di Pisa, 1998.

[LL10] J. Liu and H. Lin. Proof system for applied pi calculus. In *Proc. IFIP TCS*, volume 323 of *IFIP Advances in Inf. and Comm. Technol.*, pages 229–243. Springer, 2010.

[LV03] C. Laneve and B. Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13(5):657–683, 2003.

[Mad14] J.-M. Madiot. Coq proof scripts for some results on pi with preorders, 2014. Available at `http://madiot.org/pip-relations.tar.gz`.

[Mer00] M. Merro. *Locality in the pi-calculus and applications to distributed objects*. PhD thesis, École des Mines, France, 2000.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil90] R. Milner. Functions as processes. In *Proc. of ICALP*, LNCS, pages 167–180. Springer, 1990.

[Mil92] R. Milner. The polyadic pi-calculus (abstract). In Rance Cleaveland, editor, *Proc. of CONCUR*, volume 630 of *Lecture Notes in Computer Science*, page 1. Springer, 1992.

[Mil93] R. Milner. *The polyadic π-calculus: a tutorial*. Springer, 1993.

[MN05] M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *J. ACM*, 52(6):961–1023, 2005.

[MPS14] J.M. Madiot, D. Pous, and D. Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In Paolo Baldan and Daniele Gorla, editors, *Proc. of CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2014.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In Werner Kuich, editor, *Proc. of ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[MS04] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004.

[Nes97] U. Nestmann. What is a 'good' encoding of guarded choice? *Electronic Notes in Theoretical Computer Science*, 7(0):206 – 226, 1997. EXPRESS'97.

[Pal97] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proc. of POPL*, POPL '97, pages 256–265, New York, NY, USA, 1997. ACM.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[Pou08] D. Pous. *Techniques modulo pour les bisimulations*. Phd thesis, École Normale Supérieure de Lyon, February 2008.

[PP14] J.Å. Pohjola and J. Parrow. Bisimulation up-to techniques for psi-calculi. Draft, 2014.

[PS93] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *Logic in Computer Science, 1993. LICS '93., Proceedings of Eighth Annual IEEE Symposium on*, pages 376–385, Jun 1993.

[PS95] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Inf. Comput.*, 120(2):174–197, 1995.

[PS96] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[PS12] D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

[PV97] J. Parrow and B. Victor. The update calculus. In Michael Johnson, editor, *Proc. of AMAST'97*, volume 1349 of *LNCS*, pages 409–423. Springer, 1997. Full version available as Technical report DoCS 97/93, Uppsala University.

[PV98a] J. Parrow and B. Victor. The fusion calculus: expressiveness and symmetry in mobile processes. In *LICS*, pages 176 –185. IEEE, 1998.

[PV98b] J. Parrow and B. Victor. The tau-laws of fusion. In *CONCUR*, volume 1466 of *LNCS*, pages 99–114. Springer, 1998.

[RBR13] J. Rot, M. Bonsangue, and J. Rutten. Coalgebraic bisimulation-up-to. In *SOFSEM'13*, volume 7741 of *LNCS*, pages 369–381. Springer, 2013.

[San96a] D. Sangiorgi. Bisimulation for higher-order process calculi. *Inf. Comput.*, 131(2):141–178, 1996.

[San96b] D. Sangiorgi. Pi-calculus, internal mobility, and agent-passing calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996.

[San98] D. Sangiorgi. On the bisimulation proof method. *J. of MSCS*, 8:447–479, 1998.

[SKS11] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.

[SP07a] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. *Theor. Comput. Sci.*, 375(1-3):169–192, 2007.

[SP07b] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *J. ACM*, 54(5), 2007.

[SW01] D. Sangiorgi and D. Walker. *The Pi-Calculus: a theory of mobile processes*. Cambridge University Press, 2001.

[Tur96] N.D. Turner. *The polymorphic pi-calculus: Theory and Implementation*. PhD thesis, Department of Computer Science, University of Edinburgh, 1996.

[Vas09] V. T. Vasconcelos. Fundamentals of session types. In *Proc. of SFM*, volume 5569 of *LNCS*, pages 158–186. Springer, 2009.

[vBV09] S. van Bakel and M. G. Vigliotti. A logical interpretation of the λ-calculus into the π-calculus, preserving spine reduction and types. In *Proc. of CONCUR*, volume 5710 of *LNCS*, pages 84–98. Springer, 2009.

[vBV10] S. van Bakel and M. G. Vigliotti. Implicative logic based encoding of the λ-calculus into the π-calculus, 2010. From http://www.doc.ic.ac.uk/ svb/.

[WG04] L. Wischik and P. Gardner. Strong bisimulation for the explicit fusion calculus. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004*, volume 2987 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 2004.

[Wis01] L. Wischik. *Explicit Fusions: Theory and Implementation*. PhD thesis, Computer Laboratory, University of Cambridge, 2001.